Master's thesis

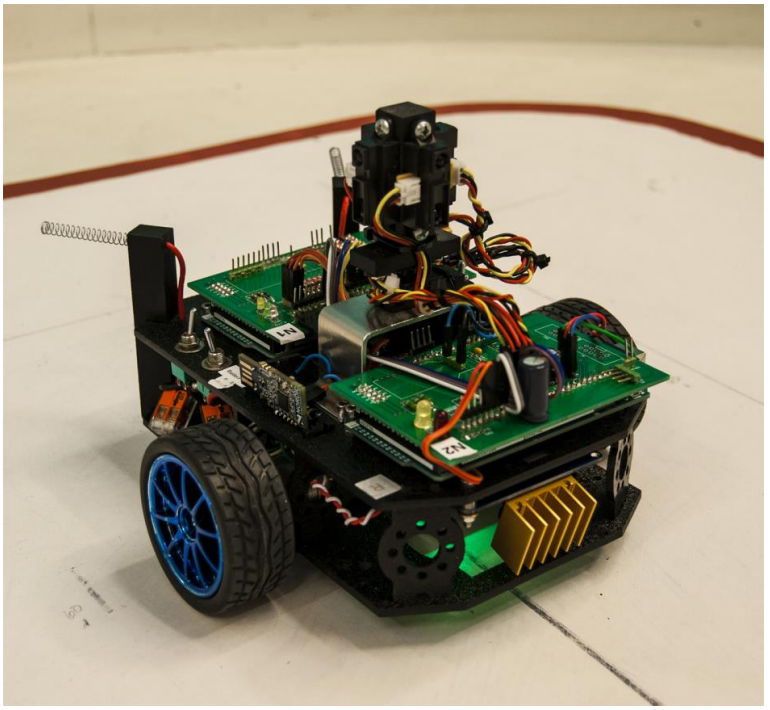
Torgeir Myrvang

Embedded design for autonomous control of diff-drive mobile robot

Master's thesis in Cybernetics and Robotics

Supervisors: Tor Onshus

Trondheim, January 2021



NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Torgeir Myrvang

Embedded design for autonomous control of diff-drive mobile robot

Master's thesis in Cybernetics and Robotics
Supervisors: Tor Onshus

Trondheim, January 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Innovation and Creativity

Problem Description

The objective for this thesis is to investigate how theoretical methods and techniques can be utilized to find practical implementation for providing autonomous control for the Arduino IR robot. The problem description for this thesis is based on the issues and proposed further work from a previous specialization project for the Arduino IR robot [19]. The objectives is summarized in the following bullet points:

- Investigate inexpensive methods for improving the computational performance of the robot by dividing the workload over several identical micro controllers, as well as, providing parallel capabilities
- Provide a foundation for autonomous control by improving the position estimation of the robot by investigate methods for filtering and processing of the sensor data
- Improve the motion of the robot by implement speed control for each wheel and investigate the use of state feedback and LQR for motion control.
- Develop procedure for path planning and object avoidance on the robot based on the IR sensors and artificial gravitational field
- Furbish the software system found in Arduino IR V1 and improving support functionalities such as the server communication

Summary and conclusion

The thesis deals with finding practical implementation for providing autonomous control for the Arduino IR robot and based on the foundation made in the specialisation project. This was carried out by improving several aspect the robot and implementing new functionalities. The improvements dealt with topics such as, hardware changes, system response, position estimation and server communication. The new functionalities includes; path planning running on the robot, motion control and support tasks.

Hardware changes included replacing the servo, IMU, encoders and motors found on the original robot, on top off, extending the computational capabilities of the robot by adding another MCU to the system. Drivers for the IMU and encoders was implemented, alongside, designing a node communication protocol for providing the link between the two MCU's. The server communication was improved, in addition to, implementation of support task for coordinating and providing virtual layer between logical nodes.

The position estimate of the robot was improved by integrating the accelerometer, alongside, implementing static filtering of raw measurement using the sample mean and variance. The heading estimation was further improved by implementing Kalman filtering on the raw measurement from the gyroscope. The motion controller was implemented as a cascaded system using a reference feed forward state feedback LQR controller for guiding the robot to a target coordinate and two PI controllers for each wheel. The LQR implementation was made possible by formulating a linear dynamic model of the robot. A path planner procedure was implemented on the robot for making it capable of navigating the environment and less dependable on the server. The implemented path planning procedure applies artificial potential field for navigation. The procedure utilizes the IR sensors for

consecutive constructing scan fields and several features of the environment in a weighted sum for finding augmented target set-point coordinates.

The integration of another MCU was successful and extended the memory, IO and computational capabilities of the robot. The node communication link provided fast and reliable transfer of data. The average deviation in the position estimated was found to be in the range of $10 - 15$ millimeters and ± 0.75 degree in the heading estimation. The use of reference feed forward state feedback LQR controller was found to give mayor improvements comparing to the old PID implementation. The controller provided fast and smooth response, in addition to tight control of the position of the robot. The straight-line deviation for the controller was in the range of $15 - 20$ millimeters. The path planner procedure was found successful in navigating a static environment given a sensible target set-point coordinate. The path planner procedure showed best result when it was used with the navigation on the server.

Oppsummering og konklusjon

Oppgaven tar for seg å finne praktisk implementering for å gi autonom kontroll av Arduino IR-roboten og er basert på grunnlaget lagt av fordypningsprosjektet. Dette ble utført ved å forbedre flere aspekter av roboten i tillegg til implementasjon nye funksjoner. Forbedringene tok for seg emner som sånn, maskinvareendringer, systemrespons, posisjonsestimering og serverkommunikasjon. De nye funksjonene inkluderer; navigeringsprosedyre utført av roboten, posisjonskontroll og andre støtteoppgaver.

Maskinvareendringer inkluderte erstatning av servo, IMU, enkodere og motorer på den opprinnelige roboten, på toppen av, og utvidet prosesseringsytelsen til roboten ved å innføre en ny MCU i systemet. Drivere for IMU og enkodere ble implementert, i tillegg til, design av en node kommunikasjonsprotokoll for å logisk koble sammen MCU-ene. Serverkommunikasjonen ble forbedret, i tillegg til implementering av støtteoppgave for koordinering av oppgaver og virtuelt lag mellom logiske noder.

Posisjonsestimatet til roboten ble forbedret ved å integrere akselerometeret inn i estimatet og implementere statisk filtrering av rå signaler ved å benytte beregnet gjennomsnitt og varians. Vinkelestimatet ble videre forbedret ved å implementere Kalman- filtrering av rå-målingene til gyroskopet. Posisjonskontrolleren er designet som et kaskadesystem ved hjelp av referanse foroverkobling og tilstandstilbakekoblet LQR-kontroller for å styre roboten til ett referanse-målkoordinat og to hastighetsregulerte PI-kontrollere for hvert hjul. Bruken av LQR ble oppnådd ved å finne en lineær dynamisk modell av roboten basert på lineær -og vinkel hastigheten. En ruteplanleggingsprosedyre ble implementert på roboten for å gjøre den i stand til å navigere i et miljø og gjøre den mindre avhengig av serveren. Den implementerte ruteplanleggingsprosedyren bruker kunstig potensielt felt for

navigering. Prosedyren bruker IR-sensorene for fortløpende konstruksjon av et skannefelt. Egenskaper fra feltet er bruk i en vektet sum for å finne ett augmentert referanse-målkoordinat.

Integrasjonen av en annen mikrokontroller var vellykket og utvidet minne, IO og prosesseringsytelsen til roboten. Node kommunikasjonsprotokollen ga rask og pålitelig overføring av data. Avviket i posisjonsestimatet ble funnet å være i området $0 - 15$ millimeter og ± 0.75 grader i vinkelestimatet. Bruken av referanse foroverkobling og tilstandstilbakekoblet LQR-kontroller gi merkbare sammenlignet med den gamle PID-implementeringen. Kontrolleren ga rask og jevn respons, i tillegg til stram regulering av posisjonen til roboten. Rettlinje avviket var i området $15 - 20$ millimeter. Navigering av roboten i ett statisk miljø et med bruk av ruteplanleggingsprosedyren var vellykket med bruk av fornuftige referanse-målkoordinater. Ruteplanleggingsprosedyren viste seg å gi best resultat når den ble brukt i kombinasjon med navigeringen på serveren.

Preface

This report act as standalone document describing the development and research on the Arduino IR robot for the Master's thesis associated with the final 30 credits course TTK4900-Engineering Cybernetics. The Master's thesis is the final course in the the MITK 2-year Master's Degree Programme in Cybernetics and robotics. The thesis is given by supervisor Tor Onshus.

Provided resources at the beginning of the thesis was the Arduino IR robot, the related source code and Atmel ICE debugging tool. For equipment, a workstation, table with test track and oscilloscope for measurements was provided. In addition, access to the motion tracking lab in B333, Campus NTNU, *Gløshaugen* and access to the ITK equipment storeroom and electronic workshop, Campus NTNU, *Gløshaugen*. The workshop provided parts and equipment for necessary hardware changes. Academic literature and sources was provided by the accumulation of various courses.

Special thanks to Tor Onshus for providing source material, vital guidance and consulting during the project. A thanks also goes to the people running the ITK workshop and fellow students in 313b for providing discussion and useful information, as well as, those who ordered parts.

Torgeir Myrvang - Trondheim Januray 2021

Contents

List of Tables

List of Figures

List of Symbols

List of Symbols

ADC	Analog to Digital Converter
ARQ	Automatic repeat request
BLE	Bluetooth Low Energy
CARE	Continuous Algebraic Riccati Equation
DARE	Discrete Algebraic Riccati Equation
DC	Direct Current
FreeRTOS	Real-time kernel
IR	Infrared
IMU	Inertial Measurement Unit
I2C	Inter-Integrated Circuit
LQR	Linear Quadratic Regulator
MCU	Micro Controller Unit
MIMO	Milti-input multi-output system
PID	Proportional, Integral and Derivative control
PWM	Pulse width modulation
RISC	Reduced Instruction Set Computer
SLAM	Simultaneous localization and mapping
SPI	Serial Peripheral Interface
USART	Universal Synchronous and Asynchronous serial Receiver and Transmitter

Δt	sampling time
u_k	Control output at iteration k
e_k	Error at iteration k
$w_{u,e}$	discrete PID weights
$K_{P,I,D}$	PID gains
$\mathbf{x}_{sp}$	Target set point coordinate
θ_r	Heading of the robot
θ_{at}	Angle to target
d_{dt}	Distance to target
PID_a	PID heading controller
PID_d	PID distance controller
$T_{inc,rec}$	intermediate input torque
$\mathrm{Dir}_{Left,Right}$	Direction for left and right motor
$T_{Left,Right}$	Input torque for left and right motor
FO, BA	Forward and backward direction
F_l, F_r	Force left and right wheel
ω_k	sampled angular velocity
v_k	sampled linear velocity
$\omega_{g_{z,y,x}}$	Scaled angular velocity IMU
$a_{z,y,x}$	Scaled linear acceleration IMU
r_g	Sampled raw gyro measurement
r_a	Sampled raw accel measurement
$F_{SF,g}$	sensitivity scaling gyro
$F_{SF,a}$	sensitivity scaling accel
o_g	scaling offset
w	Noise
r_T	Threshold rejection
μ_r	Sample mean
σ^2	Variance
$X[n]$	Random signal
$[x_{W,k}, y_{W,k}, z_{W,k}]^T$	World frame coordinates
$\mathbf{p}_k^W$	World frame position
L_k	Linear position
c_i, r_i	internal frames
$C_{l,r,k}$	sampled left and right tick speed
K_{tw}	Tick scaling factor

m_r	Mass of robot
J_r	Moment of inertia of the robot
r_{wb}	Wheel base radius
T_c	Coulomb friction
B	Coefficient of the viscous friction
μ_f	Friction coefficient
$\mathbf{K}$, $\mathbf{K}_r$	LQR gain matrices
$\mathbf{A}$, $\mathbf{B}$	system matrices
$\delta_{thrs,d}$	Region threshold
$\mathbf{r}$	Reference state
s_{ramp}	Ramp step size
$\propto$	Scaled intensity
L_{unit}	IR uint gain scale
K_{IR}	IR Correction gain
$\mathbf{F}$	Scan field
R_s	Scan resolution
θ_s	Servo angle
$\mathbf{\Gamma}$	Normalized IR measurement vector
$\mathbf{\Theta}$	Angle weight vector
$\mathbf{P}$	Proxy vector
$\mathbf{z}$	Weighted sum vector
θ_{ssa}	Scan step angle
Δ_{awd}	Angle weight direction
R_{dsi}	Sample point sector
$\theta_{aug,min}$	Augmented angle
j_{hfi}	Heading reflection index
d_{aug}	Augmented distance
$K_{\theta,rep}$	Repulsive angle gain
$K_{dist,rep}$	Repulsive distance gain
j_p	Pivot index

Chapter 1

Introduction

1.1 Background for the thesis

The project is part of a collective collaboration under the nickname *The Lego project* that dates back to 2004 and is given by supervisor Tor Onshus. As of writing this report, the collective consist of three small robots; The Arduino IR robot, The Arduino LiDAR robot and The nRF52 robot.

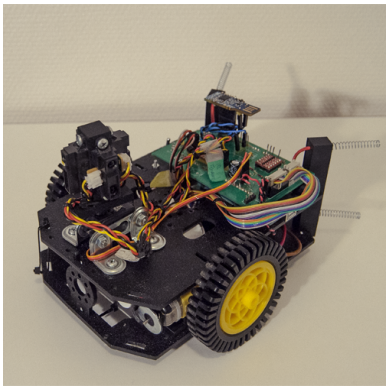


Figure 1.1: Picture of the Arduino IR robot. Myrvang (Project 2020) [19]

.

The Master thesis is based upon a specialization project carried out Spring 2020, Myrvang (Project 2020) on the Arduino IR robot [19]. The robot can be categorized as a battery operated differential drive mobile robot. It uses an embedded MCU *micro controller* of the type Atmega2560 for main logic control and communicate wirelesses with a server over BLE *Bluetooth Low Energy*. There have been several previous students that have contributed in some for or another to the Arduino IR robot. The main contributes can said to be Ese (Master 2016) [8], Andersen & Rødseth (Master 2016) [24], Lien (Master 2017) [13] and Nilssen (Master 2018) [20]. fig. 1.1 shows a picture of the robot as it was at the beginning of the thesis. Detailed description of the system will be given in chapter 2. During the the specialization project several software and hardware issues were discovered on the Arduino IR robot [19]. The project dealt with fixing these issues in addition

to lay the ground work for further improvements on the system. The system from the specialization project will hereby be referred to as Arduino IR V1 and includes the software and hardware as it was at the final stage of the project. The robot and implemented system carried out during this thesis will be referred to as Arduino IR V2. Arduino IR is refereed to as a joint designation for current and past software.

1.2 Problem description

The problem description for this thesis is based on the conclusion and proposed further work from the specialization project. The project concluded that the robot was in most part operational [19]. However there was still pending issues and limitations. The following bullet points can be said to be the main limitation within the hardware of the robot:

- The encoders have only one signal pin and cannot give information of the direction of the motors. The hall effect sensor found on the encoders are particular sensitive to external influences as they are not encapsulated. [19].
- The DC motors used in the robot suffer from wear and being underpowered. This result in large difference between the left and right motors and greatly impacts the performance of the robot.
- The servo used for controlling the servo tower is broken and need to be replaced. [19]

The software of the Arduino IR V1 have the following limitations:

- The use of PID controllers gives adequate control but suffer from large arcs and oscillatory behaviour. The "black box" design does not take into account the dynamics of the system. This makes it harder to tune the system and dynamical properties are not exploited for providing optimal control.
- There are no methods for filtering the angular velocity measured from the IMU. The nRF robot have support for Kalman filtering and is of interest to implement it on the Arduino IR.
- The non-linearity of the IR sensor is not taken into account when using the IR sensor.
- The server task running on the robot is not fully reliable and have problem syncing outgoing massages to the server, in addition to no support for sending update messages.
- The robot is entirely dependent on the main server for navigating an environment. The server have shown unstable behaviour and the robot cannot operate if the server freezes or crashes.

The objective of this master thesis is to address these issues, along side implement

solution for the following points described as further work in Myrvang (project 2020) [19]:

- Finding a dynamic model of the robot
- Collision detection and path planning
- Distributed system

Formulations of objectives

The thesis have several objective which can be summarized in the following bullet points:

- Furbish the software system found in Arduino IR V1 and improving support functionalities such as the communication with the main server.
- Investigate inexpensive methods for improving the computational performance of the robot by dividing the workload over several identical micro controllers, as well as, providing true parallel capabilities.
- Provide a foundation for autonomous control by improving the position estimation of the robot by investigate methods for filtering and processing of the sensor data.
- Further improve the position control of the robot by implement speed control for each wheel and investigate the use of state feedback and LQR.
- Develop procedure for path planning and object avoidance on the robot using the IR sensors and artificial gravitational field.

The objective of the master thesis can be summarized as:
Investigate how theoretical methods and techniques can be utilized to find practical implementation for providing autonomous control to an embedded mobile robot.

1.3 Project scope and report structure

The project focuses on improving and extending the functionalities for the Arduino IR robot.This report can be divided into the following chapters

- chapter 1: Introduction
- chapter 2: Background on the hardware and software of the Arduino IR V1, as well as, an assessment of the system.
- chapter 3: Hardware changes and soft devices. Changes made to the robot for forming the structure of Arduino IR V2.
- chapter 4: Task implementation and architecture. Description of the task architecture and support tasks running on Arduino IR V2.
- chapter 5: Improvements of the position estimation. Describe the improvements and additions made for the position estimate.
- chapter 6: Improvements of the motion control of the robot. Describe the implementation of mayor changes made to the motion control of the robot.

- chapter 7: Implementation of a path planning procedure running on the robot. Describe the developed method and task implementation for enabling navigation on the robot.
- chapter 8: Discussion and further work.

Any theory used or necessary to describe an implementation or improvements are provided at the start of respective chapter. To clearly differentiate between own development and sources, a citation on the form of *[number]* is given after a equation or statement taken from a source.

1.4 Tools used

- The software running on the robot is implemented using C and compiled using AVR gcc. FreeRTOS API [17] is used for enabling concurrent task operation on the MCU.

- The code is developed using Atmel Studio 7.0 by Microchip on a host computer and the code is downloaded to the robot using an Atmel-ICE debugger.

- Processing and visualization of result data is carried out by using MATLAB.

- Optitrack - Motion Capture system was used to record the real life behaviour of the robot. The position is recorded by placing markers on the robot.

- The report is written in LaTeX.

Chapter 2

Background

This section is dedicated to give an overview of both the hardware and software system for the Arduio IR V1. The chapter will also include a performance test and discussion of the limitation of the Arduio IR V1 design. This is meant as a motivation and bringing into context the solution described in further chapters. The chapter can be divided into two parts.

1. System description
    - Main server section 2.1
    - Hardware section 2.2
    - Software section 2.3
2. Assessment of Arduino IR V1 section 2.4

2.1 Main server

There exist to implementation of the main server; a Java server and a Thread server. The Java server is written in Java and was first developed by Andersen & Rødseth (Master 2016) [24]. Similarly, the Thread server is written in C++ and the latest development was by Mullins (Master 2020) [18]. The Main server refers to the Java server in this report, as it was exclusively used during the project.

Java Server

The Java server provides a GUI *Graphical Interfaces* for controlling the robot and navigating the robot through an environment. A nRF51 Bluetooth dongle provides wireless communication between the server and the robot. The server also provide a manual mode where target set-points coordinates can be manually issued to the robot. The server utilize the sensor readings from the robot to construct a grid map of the given environment. However, the robot needs to perform the

scaling or processing of the raw IR measurement. The server provides a limited support for collision handling of the robot. The collision handler and navigation controller was developed by Thon (Master 2016), where the navigation unit is implemented using a grid map based A* algorithm [28]. The communication with the server is acknowledge based and have several types of messages defined, where both the server and the robot agree on the meaning of the messages. The main communication stack was developed by Lien (Master 2017)[13]. The messages is defined as:

- Handshake: is sent from the robot to the server containing ID, name and parameters for the robot.
- Confirm: is sent from the server confirming that the robot is successfully connected to the server.
- Finish: is sent from the server when it is disconnected.
- Order: is sent from the server containing a target set-point coordinate on the form (x_{sp}, y_{sp}).
- Update: is sent from the robot containing processed IR measurement and estimated position. The rate of which the update messages can be sent from the robot is limited to 200 ms due to the response time of the server.
- Idle: is sent from the robot indicating that it is not busy.
- Ping: is sent from the robot. However, no action is taken by the server but can be used for alive signaling.

Thread Server

The Thread server provides all of the basic functionalities provided by the Java server, in addition to have a SLAM implementation developed by Mullins (Master 2020) [18]. The nRF robot is the only robot that have the support for communicating with the Thread server, latest development by Stenset (Master 2020) [27]. Blom (Master 2020) have implemented local networking on the Thread server using a Raspberry PI as a MQTT border router and nRF52840 dongles for interfacing between the robot and the router. The nRF52840 dongle is interfaced to the nRF robot using I2C.[3]

2.2 Hardware of Arduino IR V1

The hardware of Arduino IR V1 is based on off-the-shelf parts and is built by previous students such as Andersen & Rødseth (Master 2016) [24] and Nilssen (Master 2018) [20]. The design of the Arduino IR V1 uses two DC motors with integrated encoders as the main motion actuators and interfaced by a bidirectional motor controller card of the type 2A Dual L298 H-Bridge. The encoders consist of two neodymium 8-pole magnets and two hall-effect sensors. Further the robot have a six degrees of freedom IMU *inertial measuring unit* of the type LSM6DS3

for measuring linear acceleration and angular velocity in the x, y, z-axis [16]. In addition, the robot have a senor tower compromising of four Sharp G2D12 IR *Infrared* proximity sensors mounted on top of a PWM *Pulse Width Modulation* controlled servo. A nRF51422 dongle is used to provide wireless communication in the form of BLE 2.4 GHz [21]. The main logic controller is a 8-bit AVR RISC-based Atmega2560 microcontroller integrated on a Arduino Mega SoC with external 16 MHz clock [1]. An extension card made by Nilseen (Master 2018) [20] is mounted on top of the Arduino Mega SoC for interfacing of the various components. The main power supply of the robot is a 11.1 volt lithium-ion battery. fig. 2.1 gives a simplified overview over the system structure of Arduino IR V1 and the interfacing between the components.

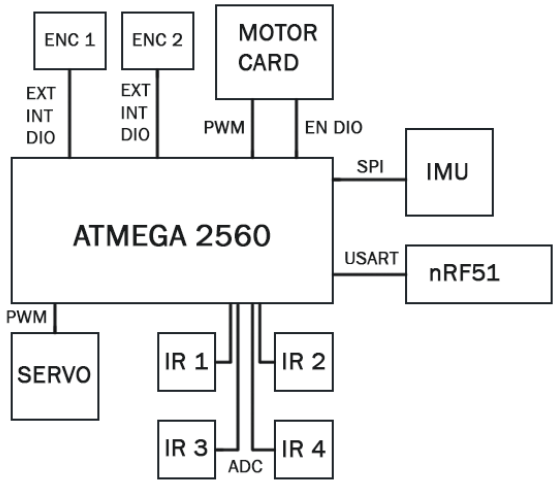


Figure 2.1: Overview of the main components used in the robot. Illustration Myrvang 2020 [19]

.

The Atmega2560 controls the speed of the two DC-motors by modulating the duty cycle of a PWM pulse generated by a built in timer on the MCU. The direction of each motor is controlled by enable signals generated by GPIO-pins *General Purpose Input/Output*. The two DC-motors have builtin gear reduction for providing a higher torque to each wheel.[20]. The angular velocity of each wheel is measured by the integrated encoders, which generates pulses that are counted using external interrupts pins on the MCU. The encoders only have one signal line each and therefore only provide limited information by the state of the wheels. The Atmega2560 fetches data from the IMU senor using a Master/Slave SPI communication. The

IR-sensors outputs a voltage corresponding to a detected distance[25] which is read using internal ADC units on the MCU. The servo is controlled by modulation the duty cycle of a PWM signal in a similar manner as the motor controller card. The nRF51422 act as an intermediate wireless transducer for communicating with the main server, where as, communication to and from the dongle and the Atmega2560 controller is carried out by USART *Universal Synchronous and Asynchronous Receiver-Transmitter*.

2.3 Software of Arduio IR V1

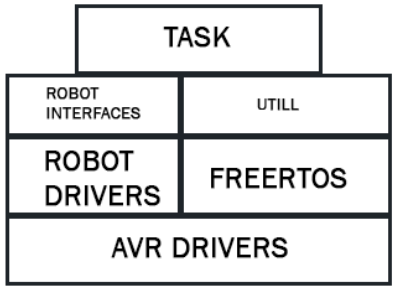


Figure 2.2: Overview of how the software running on the Arduino IR V1 is structured

The software running on the robot is written in C, using the AVR instruction set [1]. The software system uses FreeRTOS API for enabling multitasking and can be structured by abstraction layers as shown in figure 2.2. Drivers for accessing and changing registers within the Atmega2560 controller can be found on the lowest level and task implementation at the top. The system also includes drivers for interfacing the different units described in section 2.2. The robot have implemented several task, among other things, communication to and from the server, estimation of the position of the robot based on optometry from the encoders and angular velocity from the IMU, task for position control using two discrete PID controllers for driving the robot to a target set-point coordinate and a task for reading the IR-sensors and rotating the sensor tower.

Drivers and utility

The various input/output signals for interfacing the units, such as, the motor controller card, encoders and servos is implemented within the AVR driver group. This also includes the communication protocols for accessing units such as the nRF51 dongle and the IMU. These functionalities are again wrapped into robot specific interfaces, such as controlling the direction and speed of the motors, setting a specific angle for the servo, as well as, extracting data from the IMU and encoders [19].

Encoders: The pulses generated by the encoders are counted using external interrupts. The count has to be reset after each sampling and is carried out

in the task layer. [26]
Motor: The motor driver provides interface for setting the direction and speed of each of the two motors. This is carried out by inputting a *virtual* torque T_n in the range $[0\ ,\ 1000]$ that is mapped to a corresponding PWM signal. It also provide information about the direction of the motors. This is combined with the encoders in the task layer to provide information about the direction of the measured wheel speed.[26]
Servo: The servo driver provides an interface for setting the servo to a specific angle within the range of $[0, 90]$ degrees.
IR sensor: The IR sensor drivers provides functionalities for reading each of the four IR sensors. Each measurement consist of a sample mean of successive 8 ADC samples.[26]
IMU: The IMU driver is used to obtain the raw measurement of the linear acceleration and angular velocity from the IMU.[26]

The Arduino IR V1 also provides a utility library with useful functions [19]. table 2.1 list important functions that will be used and referenced further in this report.

Discrete low pass filter	Provides filtering for sampled signals tuned with the weight w
Discrete PID	A discrete PID implementation with the use of bilinear transform
LinMap	Linear mapping of an input in the range $[in_{min}, in_{max}]$ to an output in the range $[Out_{min}, Out_{max}]$ based on the point slope linear equation
Set/Rest Latch	Software Set/Reset latch used for flag operations
Stack	Interface for pushing and popping integer sized keys onto a Stack with a predefined sized array
FIFO Queue	Interface for enqueuing and dequeuing integer sized keys using a predefined sized array

Table 2.1: utility library

The discrete low pass filter uses eq. (2.1), the LinMap function uses eq. (2.2) and the output from the discrete PID controller is defined by eq. (2.3), where the weights $w_{u1,2}$ and $w_{e1,2,3}$ are based on the gains K_p, K_I, K_D [19].

$$y = (1 - w)x_{k-1} + wx_k \tag{2.1}$$

$$\text{Out} = \frac{Out_{max} - Out_{min}}{in_{max} - in_{min}}(\text{in} - in_{min}) + Out_{min} \tag{2.2}$$

$$u_k = w_{u1}u_{k-1} + w_{u2}u_{k-2} + w_{e1}e_k + w_{e2}e_{k-1} + w_{e3}e_{k-2} \tag{2.3}$$

Task implementation

The Arduino IR V1 have various task for providing and running the main functionalities of the system. The system consist of five tasks; System monitor task, Server communication task, Position controller task, Position Estimation task and Sensor tower task. All task runs periodically with the period listed in table 2.2. Data is shared using FIFO queues and task notification is used for synchronizing and signaling between the task, both provided by the FreeRTOS API.

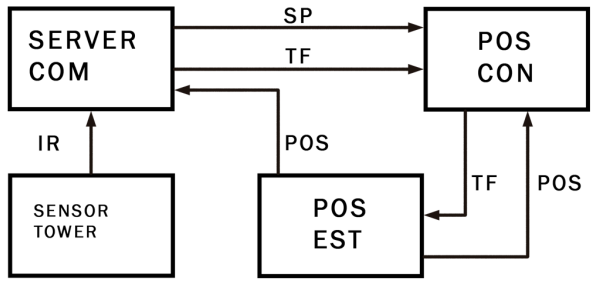


Figure 2.3: Flow diagram of the system running on Ardunio IR V1. Illustration: Myrvang 2020 [19]

.

Task name	**Priority**	**Scan time** [ms]
Sensor tower task	2	135
Server Communication task	3	485
Position estimation task	5	25
Position control task	4	50
System Monitor task	10	3940

Table 2.2: Task scan time and priority

Figure 2.3 shows the communication and signaling between the tasks, where TF means *Task notification*, SP *Set-point target coordinate* and POS *Position estimate*. The task notification from the server task to the position controller task is used to stop the robot when disconnected from the server. The position controller task uses task notification to request a new position update from the position estimation task.

Sensor tower task

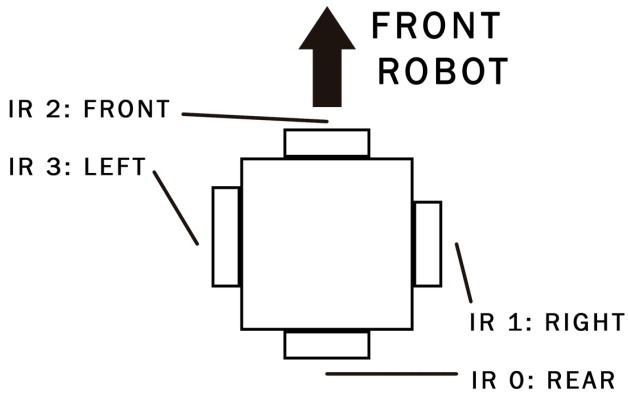


Figure 2.4: Layout of the IR sensor tower. Illustration: Myrvang 2020

.

The Sensor tower task is responsible for rotating the sensor tower and sampling the IR measurement from the four IR sensors. Each sample from the IR sensors is filtered using a discrete low pass filter. fig. 2.4 shows the placement and the naming for each IR sensor.

Server Communication task

The Arduino IR V1 have a partial implemented task for communication with the main server. It can handle the reception of orders, as well as, automatic connection and re-connection to the server. The communication stack can be divided into two layers:

1. nRF51 port: Handle the reception and transmission of bytes received from the server.[19] For each message encoded and decoded for transmission and reception respectively using consistent overhead byte stuffing [13].
2. Server message: Converting connection, orders and update massages to byte streams and providing means for acknowledged incoming massages.

Position estimation task

The Position estimation task is responsible for estimating the position of the robot. The Arduino IR V1 uses a combination of sampled measurement from the IMU gyroscope z-axis and sampled tick counts from the encoders to estimate the position of the robot. The position estimator uses two reference frames; a frame moving with the robot R and a world frame W stationary to the starting point of the robot. The Arduino IR V1 uses equation 2.4 for estimation the position

$\mathbf{p}_k^W = [x_{W,k}, y_{W,k}, z_{W,k}]^T$ of the robot in the world frame. $\theta_{z,k}^W$ is the current heading of the robot in the world frame and $\mathbf{p}_k^R = [x_{R,k}, y_{R,k}, z_{R,k}]^T$ is the robot current position in the R frame defined by the kinematic equation in eq. (2.5) [26].

$$\mathbf{p}_k^W = \mathbf{p}_{k-1}^W + \mathbf{R}_{\theta_{z,k}^W} \mathbf{p}_k^R \tag{2.4}$$

$$\mathbf{p}_k^R = \begin{cases} L_k \cos\theta_{R,k} \\ L_k \sin\theta_{R,k} \\ 0 \end{cases} \tag{2.5}$$

$$L_k = (K_l C_{l,k} + K_r C_{r,k})/2 \tag{2.6}$$

$$\theta_{z,k}^W = \theta_{z,k-1}^W + \theta_{R,k} \tag{2.7}$$

$$\theta_{R,k} = K_{gz} \frac{T_s}{2} (\omega_{z,k} + \omega_{z,k-1}) \tag{2.8}$$

L_k is the linear position found using eq. (2.6) for the left and right tick speed $C_{l,k}$ and $C_{r,k}$. $\theta_{R,k}$ is estimated using eq. (2.8), which is a numeric integration method found by bi-linear transformation, where $\theta_{R,k-1} = 0$, K_{gz} is constant scaling factor and ω_k is the angular velocity from the IMU. $\theta_{R,k}$ uses degrees as unit such that it can be represented by a 16-integer and therefore save memory space on the microcontroller. The heading $\theta_{z,k}^W$ is found using the relationship defined in equation 2.7. $\theta_{z,k}^W$ is in radians and $\theta_{R,k}$ is scaled accordingly [19].

$$y_{z,k} = g_z + w_z \tag{2.9}$$

$$r_{z,k} = y_{z,k} - o_f \tag{2.10}$$

$$r_{z,k} = \begin{cases} r_{z,k} r_{z,k} & > |r_T| \\ 0 r_{z,k} & < |r_T| \end{cases} \tag{2.11}$$

$$r_{z,k} = (r_{z,k} + r_{z,k-1})/2 \tag{2.12}$$

Arduino IR V1 assumes the relationship in equation 2.9 when processing the measured raw value from the IMU gyroscope z-axis. $y_{z,k}$ is the sampled measurement from IMU gyroscope z-axis, g_z is the true raw value, w_z is additive noise [19]. For avoiding drifting in $\theta_{R,k}$, when the robot is stationary, $r_{z,k}$ is processed sequentially using equation 2.10, 2.11 and 2.12. o_f is a constant value found empirically to shift $r_{z,k}$ closer to zero. $r_{z,k}$ is further processed by rejecting any value below a certain threshold r_T. When the robot is moving $r_{z,k}$ is processed by a simple sample mean of two.

Position control task

The position controller task is responsible for driving the robot to a specified set point coordinate defined by $\mathbf{x}_{sp} = [x_{sp}, y_{sp}]^T$ using a position controller. The

controller uses a notion of attractive field for the error between the heading of the robot θ_r and the angle to target θ_{at}, as well as, the current traveled distance of the robot and the distance to target using the following equations:

$$d_r = ||\mathrm{x}_r|| \tag{2.13}$$
$$d_{dt} = ||\mathrm{x}_s p|| \tag{2.14}$$
$$d_{dt} = d_{sp} - d_r \tag{2.15}$$
$$\theta_{at} = \mathrm{atan2}((y_{sp} - y_r), (x_{sp} - x_r)) \tag{2.16}$$
$$\theta_d = \theta_{at} - \theta_r \tag{2.17}$$
$$e_d = -d_{dt} \tag{2.18}$$
$$e_a = \theta_d \tag{2.19}$$

The controller uses a set of two discrete PID controllers PID_a and PID_d defined by eq. (2.3) [19].

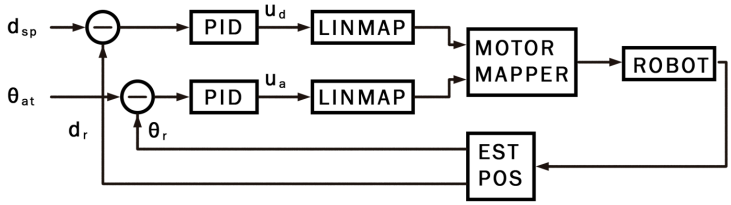


Figure 2.5: Block diagram of the controller structure for Arduino IR V1. Illustration: Myrvang 2020 [19]

.

The controller PID_a tries to minimize the deviation between the heading of the robot θ_r and the angle to target θ_{at}. Likewise, the controller PID_d tries to minimize the deviation between the traveled distance of the robot d_r and the distance to target d_{dt}. Figure 2.5 shows a block diagram of the controller structure. The output of the two controllers are scaled using the LinMap function eq. (2.2). The scaled values are feed to the open loop motor mapper, that maps the controller outputs to a corresponding input torque and direction. The motor controller uses a form of fussy logic based on the values of the control output. The interface takes in two inputs u_x and u_y The speed for each wheel is calculated by the intermediate values

T_{inc} and T_{rec} defined in eq. (2.20).

$$\begin{bmatrix} T_{inc} \\ T_{rec} \end{bmatrix} = \begin{cases} \begin{bmatrix} u_x \\ u_x \end{bmatrix} & u_x > I \\ \begin{bmatrix} u_y + u_x \\ u_y \end{bmatrix} & u_x > 400 \\ \begin{bmatrix} u_y + \frac{u_y\left(\frac{u_x}{20}\right)}{100} \\ 0 \end{bmatrix} & I > u_x > 400 \end{cases} \tag{2.20}$$

case 1:	$T_{Left} = T_{inc}$	$\mathrm{Dir}_{Left} = FO$	
	$T_{Right} = T_{rec}$	$u_x < I$: $\mathrm{Dir}_{Right} = FO$	$u_x > I$: $\mathrm{Dir}_{Right} = BA$
case 2:	$T_{Left} = T_{rec}$	$u_x < I$: $\mathrm{Dir}_{Left} = FO$	$u_x > I$: $\mathrm{Dir}_{Left} = BA$
	$T_{Right} = T_{inc}$	$\mathrm{Dir}_{Right} = FO$	
case 3:	$T_{Left} = T_{inc}$	$\mathrm{Dir}_{Left} = BA$	
	$T_{Right} = T_{rec}$	$u_x < I$: $\mathrm{Dir}_{Right} = BA$	$u_x > I$: $\mathrm{Dir}_{Right} = FO$
case 4:	$T_{Left} = T_{inc}$	$u_x < I$: $\mathrm{Dir}_{Left} = BA$	$u_x > I$: $\mathrm{Dir}_{Left} = FO$
	$T_{Right} = T_{rec}$	$\mathrm{Dir}_{Right} = BA$	

Table 2.3: The four cases for setting the virtual torque and direction for the left and right motor

Table 2.3 shows how T_{Left} and T_{Right} is set using equation 2.20, in addition to the direction for the left and right wheel, where FO and BA is the forward and backward direction respectively [26].

2.4 Assessment of Arduino IR V1

Square test

To form a baseline and motivation for the thesis, a performance test was conducted on Arduino IR V1. The particular test can be nicknamed *Square test* and consisting of driving the robot to a set of target coordinates forming a $1m \times 1m$ and a $2m \times 2m$ square using the values defined in table 2.4. The respective position controller uses a target threshold of 30 mm. fig. 2.6(a) and fig. 2.6(b) shows the resulting trajectory for Square test 1000×1000 $[mm]$ and Square test 2000×2000 $[mm]$ respectively using the hardware and software described in section 2.2 and section 2.3.

Name	SP
Square 1	$[(1000, 0), (1000, 1000), (0, 1000), (0, 0)]$ [mm]
Square 2	$[(2000, 0), (2000, 2000), (0, 2000), (0, 0)]$ [mm]

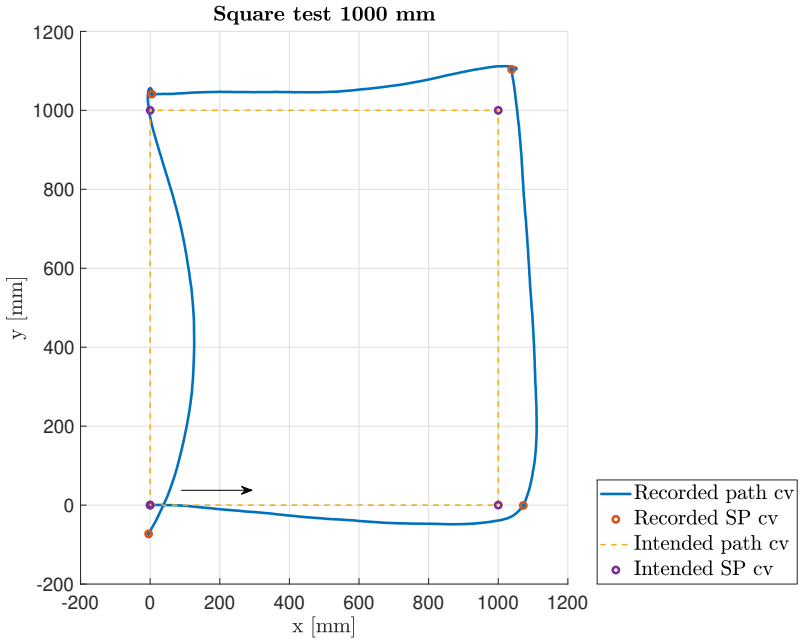
Table 2.4: Coordinate set point used in the two sqaure test for Arduino IR V1

Result

Table 2.5 shows the resulting end coordinate for each set point coordinate, which reviles a considerable large error between the recorded and indented end-point coordinate. At the same time, the deviation is consistent for each point, indicating a large, but constant error. In its basic form, the position controller used in Ardunio IR V1 has its strength of handling disturbances. From Myrvang (project 2020) it was concluded that the disturbance was produced by a considerable difference in the output torque for the two DC motors used in Arduino IR V1 [19]. The method in its purest form have the advantage that its only need a input coordinate and two controllers to get reasonable close to a target coordinate within a certain radius relative to the robot.

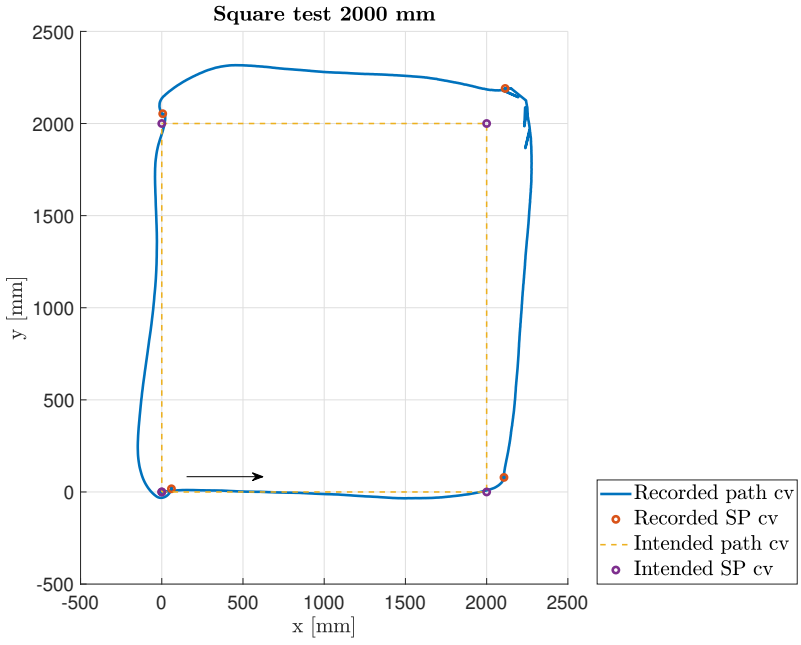
Name	Result
Square 1	$[(1072, -0.8), (1038.4, 1103.6), (4.7, 1041.1), (-4.5, -72.8)]$ [mm]
Square 2	$[(2106.8, 78.78), (2114.3, 2190.7), (5.87, 2049.9), (59.5, 17.2)]$ [mm]

Table 2.5: Recorded end points for the two square test for Arduino IR V1



(a) Result of the 1000 × 1000 mm square test for Arduino IR V1

.



(b) Result of the 2000 × 2000 mm square test for Arduino IR V1

.

However the use of PID controller can in most cases only give adequate control. The motion results in large arcs as the robot is driving to the target set-point coordinate as seen in fig. 2.6(a) and fig. 2.6(b). As concluded from Myrvang (project 2020) this is mostly due to the rotation of 90 degrees, which result in a large proportional input error and resulting large output gain. A way of combating this is to use gain scheduling for different turn angels. To the contrary this makes the controller harder to tune as one have to deal with multiple gain parameters. The controller also uses a form of "black box" design, which is one of the less positive aspect of using PID controllers for this particular task. Far more desirable would be to obtain a dynamical description of the robot and using this model to provide a more optimal control.

It was also concluded from the project that the position controller can only be as good as the position estimate of the robot. [19] There are several problems with the estimation method used in Arduino IR V1. First, the estimation is only loosely based on a kinematic equation and expresses some oddities that makes it harder to reason about the applied method. Furthermore, the position estimate also lacks proper filtering of the angular velocity obtained from the IMU. The threshold for removing noise when the robot is stationary is not based on the properties of the signal. From Myrvang (project 2020) it was observed that the noise level of the raw data increases correspondingly to the decrease in the charge of the battery. This result in the value no longer holds when the robot is used over time. The encoders are also insufficient, in that they cannot give information about the direction of each wheel. This must be handled by reading the logical state of the motor controller signal pins. This means that there can be a significant delay from the actual direction of the wheel and the observed direction.

Chapter 3

Hardware changes and soft devices

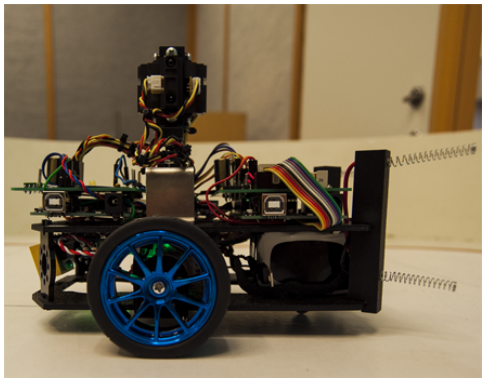
The general performance of the Arduino IR robot relies greatly, in a broader sense, on the underlying hardware. This is especially true when addressing the objectives concerning improving the position estimation and motion control. Topics that indirectly deals with improving the performance of the robot is the main theme of this chapter. This chapter will describe changes made to the robot that constitute to the foundation of implementation described in future chapters and will detail how new hardware is integrated into the system and related soft devices. The chapter can be structured in the following way

1. The new robot section 3.1
2. Soft devices section 3.2

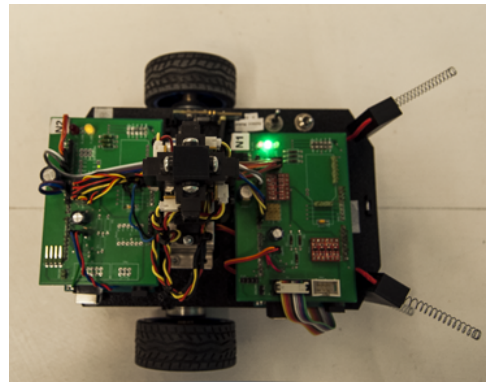
3.1 The new robot - Arduino IR V2

The robot have underwent mayor changes to its structure and components. fig. 3.1(a) and fig. 3.1(b) gives a profiled view from the side and from above of the new robot. The original LSM6DS3 6 Dof IMU found on the Arduino IR V1 has been replaced. The new robot has been equipped with an ICM 20948 9 Dof IMU. This change is made due to the later IMU deliver more functionalities in one package, in the form of gyroscope, accelerometer and magnetometer [7]. Components such as the servo and DC motors has been replaced due to being either broken or worn out, as discussed in section 2.4 and Myrvang (project 2020) [19]. The new DC motors for the Arduino IR robot is based on suggestion from Stenset (Master 2020) [27] providing wheels with a smaller diameter and wider wheel base, in addition to, higher torque for the motors as specified in table 1. This can provide better trac-

tion and reducing the moment of inertia for the wheels compared to the original wheels. The integrated encoders on the new motors have two hall-effect sensors which can be used to measure the speed and direction of rotation for the wheels as opposed to the encoders found on Arduino IR V1.



(a) Side view of the new hardware setup for Arduino IR V2.



(b) View of the Arduino IR V2 robot from above.

.

One of the objectives in section 1.2 is to design a distributed system for dividing the work load and provide parallel capabilities. For the distributed system, an embedded MCU have in general highly limited memory and computational resources but provide robust IO interfacing and low energy operation [1]. The aspect of limited resources affect all of the software on the system, but especially the tasks running on the robot. The scheduling of task becomes harder as more task are added and jittering introduced by hardware interrupts. It becomes increasingly difficult to guaranteeing real-time requirements using only one MCU. Therefore, dividing the workload over several MCU is an inexpensive method for improving the performance of the robot.

This is carried out alongside with, extending the computational capabilities of the robot, by adding another Atmega2560 microcontroller to the robot. Choosing this controller is attractive for ensuring efficient implementation of higher level functionalities, as Myrvang (project 2020) [19] dealt with providing more reliable driver functionalities for the Atmega2560 microcontroller.

Components layout

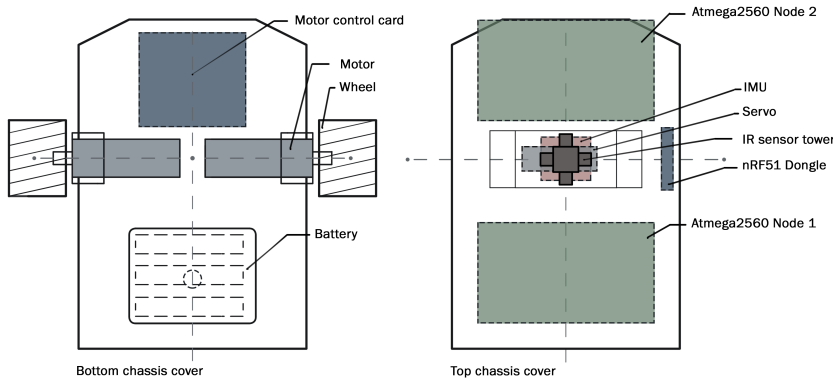


Figure 3.2: Illustration showing the layout of the components for Arduino IR V2.

In order to better accompany the new components and make the robot more rigid, the placement and arrangement of the devices used in the Arduino IR robot has also been changed. An illustration of the new layout can be seen fig. 3.2 . Both the two Atmega2560 controllers, the IMU and servo are all mounted on the top chassis frame. In Arduino IR V1, the IMU was mounted on the underside of the bottom chassis frame. In the new layout this is not preferred. The IMU is placed on the top chassis to reduce the noise induced by the motors. In addition, the servo and the IMU is mounted as close as possible to the center of rotation for the wheel axis. This advantageous when using the IMU for computing the heading of the robot and using the IR sensors for distance measurements. The IR sensor layout is the same as shown in fig. 2.4. The motor controller card is mounted on the underside of the top chassis as represented by the dotted lines in fig. 3.2. The given layout provides ease of access for the wiring to the motors, encoders and battery by flipping the top cover. The top and bottom chassis frames are held together by 7 mounts and provides a highly rigid frame for the robot. Howevet, due to limited mounting points on the robot chassis frame, the motor are mounted with a positive offset, which also apply for the battery.

System structure

Figure 3.3 shows a block diagram of the new system design using two Atmega2560 controllers in parallel. The two controllers are designated as logic Node 1 and logic Node 2. Logic node 1 is responsible for estimating the position and controlling the motion of the robot. Therefore, node 1 is used to sample the sensor reading from

the encoders and IMU, in addition to handle the control of the two DC motors. Logic node 2, as seen to the left in fig. 3.3 is responsible for interface the four IR sensors, controlling the servo and interface the nRF51 dongle. It is also responsible for handling the communication to the main server.

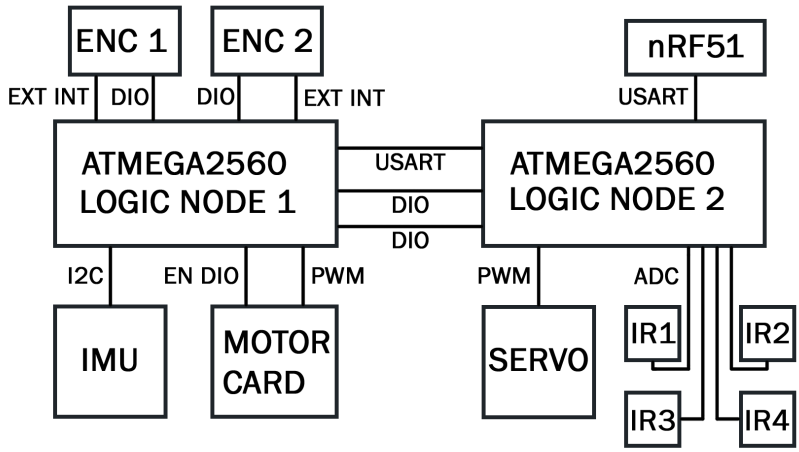


Figure 3.3: Block diagram for peripherals for Arduino IR V2.

3.2 Soft devices

IMU

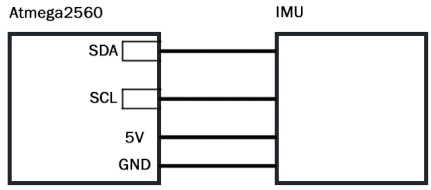


Figure 3.4: Wiring diagram between the Atmega2560 and the ICM 20948 IMU

fig. 3.4 shows how the IMU is connected to the Atmega2560 controller. I2C is used to interface the MCU to the IMU. The raw values from the 3-axis gyroscope and 3-axis accelerometer found on the IMU, have an output resolution of 16-bit and is delivered as signed 16-bit integers. The sensor measurement from the IMU

is sampled every 50 ms by the microcontroller. The values are extracted by successive reading a high and low 8-bit registers, which are concatenated to form the raw values r_g and r_a, corresponding to the angular velocity and linear acceleration. The IMU is configured with the parameters shown in table 3. The gyroscope measure the angular velocity in dps and the accelerometer linear acceleration in g [7]. To find the angular velocity and linear acceleration, eq. (3.1) and eq. (3.2) is used, where

$$\omega_{g_{z,y,x}} = \frac{r_g}{F_{SF,g} \pm o_g} \; [dps] \tag{3.1}$$

$$a_{z,y,x} = 9806 * \frac{r_a}{F_{SF,a}} \; [g] \tag{3.2}$$

$$\tag{3.3}$$

o_g is a scaling offset, $F_{SF,g}$ and $F_{SF,a}$ are sensitivity scaling factors found in table 3.

Encoders

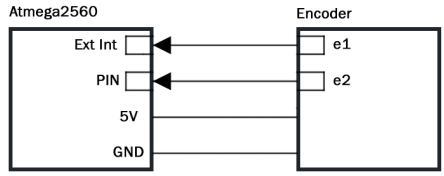


Figure 3.5: Wiring diagram between the Atmega2560 and the encoders

The two integrated encoders on the motors have two signals pins e1 and e2 each that generates pulses according to the movement of the wheel. fig. 3.5 shows how the encoders are connected to the Atmega2560 controller.

Depending on the direction off the motors, the pulse from e2 is either -90 or 90 phase shifted relative to e1. The phase shift is used to differentiate the forward and backward rotational direction of the wheel. fig. 3.6 gives an illustration of the signals generated by e1 and e2, as well as, correspondingly phase shift. The sampled raw tick speed of the left and right wheel defined as $C_{l,k}$ and $C_{r,k}$ can be found by counting the pulses, sampling the count with a sampling time Δt and resetting the count after each sample.

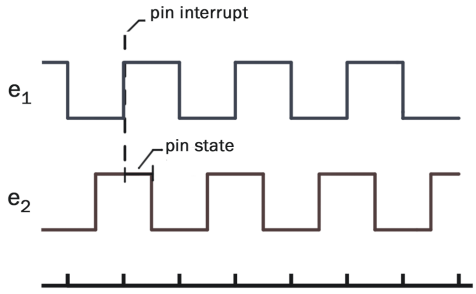


Figure 3.6: Encoder signal e1 and e2

However, the logical level between e1 and e2 can be in one of the four combination $00, 01, 10, 11$ when the a wheel stops moving. Therefore, only the pulses generated by the e1 are counted up or down depending on the logical level of e2. To achieve this, as well as, to making the driver encapsulated, two ISR *interrupt service routines* are used. An external ISR is used to count the pulses and a timer overflow ISR is used for sampling the tick speed. As illustrated in fig. 3.6, the external ISR is set to trigger on a rising edge for the signal e1. Within the ISR, the pulse count from e1 is either added or subtracted by one depending on the truth table in table 3.1 for e2. The Atmega2560 controller have only a limited number of timer prescale values used for generating the timer overflow,where the highest is 1024 [1]. As a consequence, the timer overflow ISR is set to trigger every $16.32\ ms$ and a sampling counter is used for sampling the tick speed $C_{l,k}$ and $C_{r,k}$ every $49\ ms$.

e1	1	1	add
e2	1	0	subtract

Table 3.1: Truth table for counting the encoder pulses

Within the timer overflow ISR, the sampled tick speed is copied to variables for the left and right wheel respectively and the pulse count is zeroed. The variables are global variables only visible to the scope of the driver. This means that the tick measurement is effectively seen as a read-only value to any application outside the driver and makes the read functionalities thread safe.

$$K_{tw} = \frac{\text{Ticks per wheel rev}}{2\pi} \tag{3.4}$$

$$\omega_{m,k} = \frac{1}{K_{tw}} \frac{C_k}{\Delta t} \text{ Rad/s} \tag{3.5}$$

$$v_{m,k} = K_{vw} \frac{C_k}{\Delta t} \text{ mm/s} \tag{3.6}$$

$$K_{vw} = \frac{r_w}{K_{tw}} \tag{3.7}$$

The sampled tick speed needs to be scaled such that the velocity for each wheel can be extracted. The angular velocity for each wheel can be expressed in radian

defined in eq. (3.5) per second by converting the the sampled tick speed $C_{l,k}$ and $C_{r,k}$ using the scaling factor in eq. (3.4). The linear velocity is found by using eq. (3.5), where K_{tw} is the scaling factor for scaling the ticks to angular velocity, $\omega_{m,k}$ is the sampled angular velocity for the wheels, r_w is the wheel radius and $v_{m,k}$ is the sampled velocity. The parameters in table 2 is used.

Node communication

As described in section 3.1 the new design is a distributed system with two logical units for dividing the workload, As a result, a communication link between the nodes are needed. The link is designed to provide a physical communication interface with a virtual layer, such that the two units act as one, in addition to, enabling information given by FreeRTOS queues and events to be shared between the two nodes. Another goal of the design is to transfer bytes of data with the least amount of overhead, such that the impact of the communication is minimal. The interface is designed as a USART/SPI hybrid and provides half duplex communication. The interface uses USART as basis and two GPIO-pins on the Atmega2560 controllers. The GPIO-pins have a double purpose, where they act both as control signal for message termination and synchronization when sending and receiving. fig. 3.8 shows how the two nodes are connected.

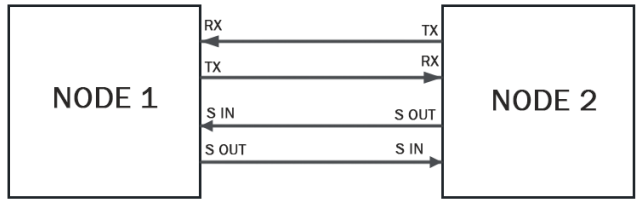


Figure 3.7: Wiring diagram for the node communication interface

The output signal pin from one node is connected to the input signal pin of the other and vice versa. The synchronization is performed by locking the USART port when transmission and receiving, hindering the node to transmit at the same time. The USART port on the two Atmega2560 controllers uses the specification stated in table 3.2

Frame format:	8 data, 2 stop bit
Baud rate:	250 Kbps or 31 KB/s

Table 3.2: USART specification for the node communication

ID (1 byte)	Data size (1 byte)	Data (0 - 255 bytes)

Table 3.3: Caption

The massage format consist of a header of two bytes and the data as shown in table 3.3. The first byte of the header is used for identifying the type or shape of the data. The next byte is the size in bytes of the variable to be transmitted. The identifier byte is used by the node participant to convert a received message to its respective variable.

ID	type
0	Ignored
1	Error ID
2-5	Events
20-255	Robot data

Table 3.4: ID specification for the node communication

table 3.4 shows the legal variables that can be used to reconstruct a received byte stream. Serial USART transmission and reception of the value zero is normally treated as a delimiter and is automatically added when sending a byte stream. In the node communication design, delimiting of a messages is carried out by the GPIO pins. Therefore, the protocol specifies that an ID with value of zero is considered illegal and will be ignored by the nodes. The interface also needs to support any type, from single variables as int and float to composed types, such as, structs. As a result, the data is converted to a serial byte stream by extracting the memory address of the variable and coping the content to a buffer which correspond to the message format in 3.3. The structure of the interface can be divided into three abstraction levels

1. Node port: Device specific for interfacing with the USART port, hardware interrupts for reception and pin signal termination.
2. Node transfer: Provides the means for converting to serial stream and transferring data, as well as, reception and data integrity check.
3. Node task: Provide the virtual layer between the nodes such that FreeRTOS queues and events can be exchanged.

The reception and transmission of a byte stream in the Node port layer is carried out by first reading the input signal pin to check if the port is busy. The protocol specifies that the port is considered busy when the input signal pin is logical high and correspondingly not busy when the pin is logical low. Second, if the port is not busy, the out signal pin is pulled high, locking the port for both participants. The first two header bytes are sent first, followed by the data bytes. Lastly, the

node releases the port after the last byte is sent by pulling the out signal pin low. The receiving node uses a ISR for capturing the data sent over the link. When the first byte is received, the receiving node pull its own out signal pin high. The falling edge of the out signal pin from the transmitting node is used as message termination. The receiving node uses an external pin ISR for detecting this change. The out signal pin from the receiving node will stay high until the data is processed. The data received is buffered into a FIFO queue. A "data ready" flag is set when the external pin ISR is triggered. fig. 3.8 gives an illustration of the reception and transmission signaling.

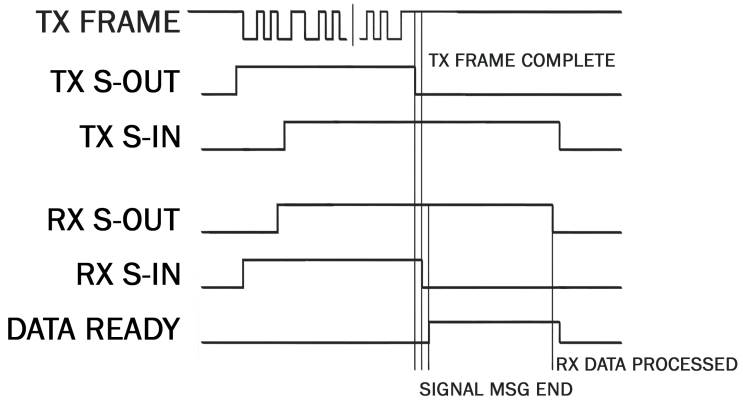


Figure 3.8: Signaling for transmission and reception

The Node transfer provides functionalities for processing outgoing and incoming messages. Listed in algorithm 1 is the handler used for transmission a byte stream. The port is locked corresponding to the output pin specified in the Node port layer. The transmission handler will set a "Transmission-Not-Success" flag if the port is used. If the port is available, the ID and Size byte is transmitted first, followed by the data bytes.Handler listed in algorithm 2 is used for message reception. When a message is received, the ID and Data size header is extracted from the FIFO queue found in the Node port layer. To check if the data was sent correctly, the Data size byte is compared with the number of elements left in the FIFO queue. If these two numbers don't mach, a received error flag is set. The receiving node will then prematurely realise the port and send an error massage to the transmitting node, containing the error ID. If the number of elements received matches with the data

size header, the data is transferred to a data buffer and the FIFO queue is flushed.

Algorithm 1: Transmission handler

```
if Port NOT busy then
    Lock port
    Transmit(ID byte)
    Transmit(Data Size Byte)
    for Data size do
        Transmit(Data Bytes)
    end for
    Release Port
else
    Set "Transmission-Not-Success" flag
end if
```

Algorithm 2: Node reception handler

```
if Data NOT Fetched then
    ID ← Dequeue(Rx ISR buffer)
    Data size ← Dequeue(Rx ISR buffer)
    if Data size = Elements in Rx buffer then
        for Data size do
            Data buffer ← Dequeue(Rx ISR buffer)
        end for
    else
        if ID = Error ID then
            Set Transmission Error
        else
            Set Reception Error
        end if
    end if
end if
```

Chapter 4

Task architecture

This chapter seek to give an overview of the complete software system implemented on the Arduino IR V2. An overview of the task architecture and communication is given, in addition to providing information about support task such as, task implementation of the Node communication protocol, complete implementation of the server communication and the coordination of the tasks. The chapter is structured into two parts:

1. Task architecture section 4.1
2. Support tasks section 4.2

4.1 Task implementation and architecture

The system design of Arduino IR V2 uses the tasks described in section 2.3 as basis. However, the new system have seen mayor changes to the original design found on Arduino IR V1. Two key aspects are the introduction of the two logic nodes and definition of new tasks. table 4.1 shows the tasks designated to each node.

Node 1	**Node 2**
System monitor task	System monitor task
Node communication task	Node communication task
Position estimation task	Server communication task
LQR control task	Coordinator task
Speed control task	Path planner task

Table 4.1: List of defined task and assignment between the nodes

Node 1 runs the tasks that are responsible for handling the robot. The estimation

task runs the code for estimation the position of the robot, described in chapter 5. The LQR control task, described in chapter 6, carries out the control for guiding the robot to a target set-point coordinate. The Speed control task handles the control for maintaining a given angular velocity for each wheel, detailed in chapter 6.

Node 2 runs the tasks that are responsible for coordinating, communication with the main server and observing the environment. The Server communication task handle the communication with the main server. The Coordinator task is used to determine and coordinate which operation the robot should perform based on request, events and data queues produced from both nodes. The Path planner task, detailed in chapter 7, is responsible for navigating the robot in an environment by computing augmented target set-point coordinate based on the observation extracted from the IR sensors.

Node 1	**Priority**	**Period** $[ms]$
System monitor task	6	1948
Node communication task	4	75
Position estimation task	5	50
LQR control task	3	60
Speed control task	4	27

Table 4.2: Priority and period for the tasks on Node 1

Node 2	**Priority**	**Period** $[ms]$
System monitor task	6	1948
Node communication task	5	75
Server communication task	3	100
Coordinator task	4	28
Path planner task	5	60

Table 4.3: Priority and period for the tasks on Node 2

All of the tasks found on both of the nodes runs periodically with the respective periods defined in table 4.2 and table 4.3. The periods and priorities are assigned based on the critically of the task and the total system performance. The periods are set to reduce the interference between the tasks. They are set to be not multiple of each other, such that the lower priority does not suffer starvation.

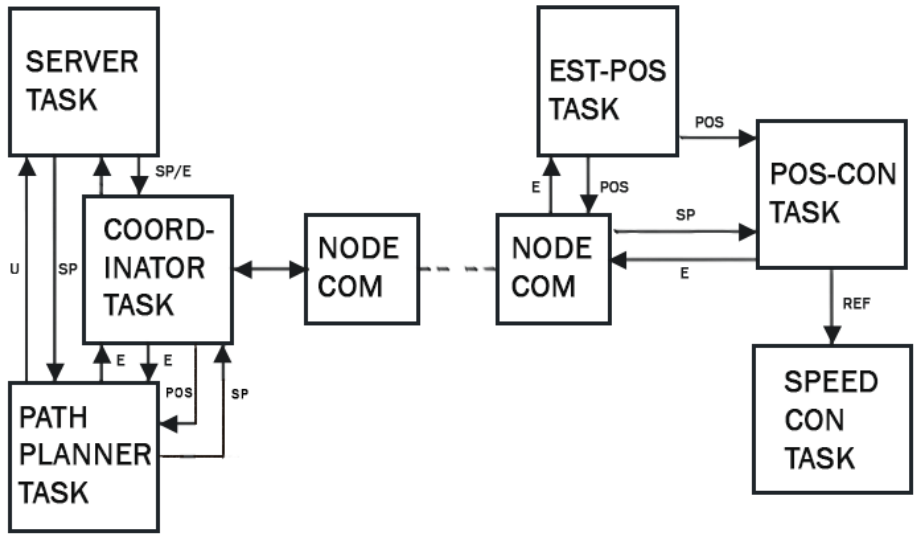


Figure 4.1: Block diagram showing the signaling between internal tasks and the two nodes.

.

Each task have a modular and encapsulated design, where a task have a set of input and output channels for which data is exchanged. Likewise for the design in Arduino IR V1, data is shared using FreeRTOS queues. FreeRTOS events are used for signaling and sending data request. The two nodes communicate with each other using the node communication task, where each node runs an identical instance of the task. The Node communication protocol, described in section 3.2, provide a virtual layer for which data can be exchanged and the node communication task act as an interface. The signaling and data exchange between internal task and the two nodes are shown in fig. 4.1. **E** stands for event/request, **SP** is a target set-point coordinate, **POS** is the position estimate, **U** is the IR sensor update and **REF** is reference input torque. Both **SP**, **POS**, **U** and **REF** are exchanged using queues.

4.2 Support Tasks

Coordinator task

The purpose of the Coordinator task is to coordinate and tie together the operation of all the other tasks. The task enables action of higher abstraction to be performed by the robot. The coordinate task acts on incoming events and data queue from the other tasks and decide appropriate action to be taken by sending out request. Several other task needs data produced and operations performed by each other. To

only transfer data when a task needs it, request flag is implemented as events. All request and events are forwarded to the coordinate task such that the robot is able to perform different actions and perform these in a asynchronous manner. Some examples are:

- Enable position controller by sending a start signal and SP.
- Align the robot to the target coordinate by sending a start request.
- Move the robot to the target coordinate by sending a start request.
- Reset the LQR controller by sending a reset request.
- Get position update by issuing a position request.
- Request the Path planner task to find out if the current target coordinate is safe to traverse or compute a augmented target.
- Request the Path planner task to perform an a scan of the environment for either sending sensor updates to the Server communication task or rerouting the robot.

Node communication task

The node communication task is used to provide a virtual layer between the nodes such that implemented task on each node can communicate with each other by using FreeRTOS queues and events. The task implementation gives a clean abstraction for hiding the complexity in the Node transfer layer and Node port layer described in section 3.2. The task is responsible for conversion between the FreeRTOS queues and events and serial byte streams. The conversion is carried out by utilizing an ID look-up table, where both nodes agree on the same ID to variable mapping. The task is designed as a state machine as illustrated in fig. 4.2.

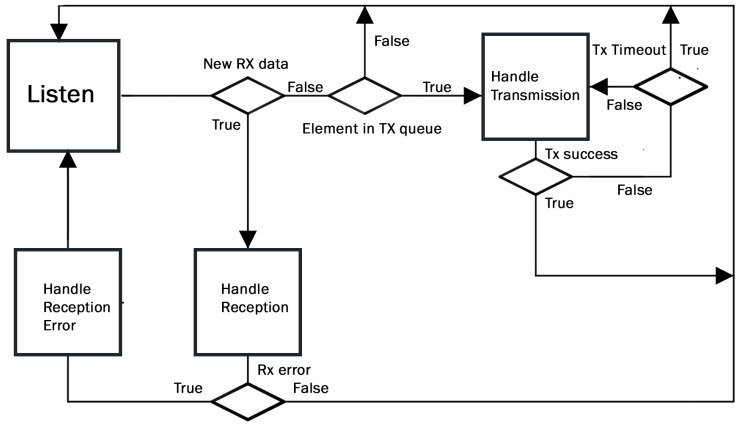


Figure 4.2: Flow chart for the node communication task

The task is set to run periodically, where the default state is the **Listen** state. Within the state, the "Data-ready" flag is polled, as well as, any newly received internal FreeRTOS events or queue updates. The ID for pending data to be sent over the node communication link is stored in a FIFO queue. If a new message is received, the task goes to the **Handle reception** state, where the receive handler listed in algorithm 3 is used. Received data, stored in the data buffer, is compared with the ID look-up table. If there is an ID for the defined variable, the content from the data buffer is copied to the respective variable. Any reception error thrown by the RX data fetcher handler listen in algorithm 2 in section 3.2 is handled by releasing the node port and sending an error signal to the transmitting node.

Algorithm 3: Node task reception handler

```
Extract buffer from RX data fetcher using algorithm 2
if NOT Reception Error then
    Fetch Data type conversion from ID look-up table
    Convert data buffer to data type
    if NOT Data Error then
        Set event if Event type
        Place data in FreeRTOS queue if data type
        Go to state Listen
    end if
else
    Handle error by release the node port and flush the reception buffer
    Send error signal to the transmitting node
end if
```

The task transition to the **Handle transmission** state listed in algorithm 4, if there is no incoming messages and the TX ID FIFO queue is populated. The task will try to send the messages by a given amount of times as shown in fig. 4.2. If the number of unsuccessful attempts exceed the Timeout condition, the respective ID is placed back into the FIFO queue and the task returns to the **Listen** state.

Algorithm 4: Node task transmission handler

```
Based on ID look-up table: Copy content from Data type to data buffer
Send data using Transmission handler in algorithm 1
if Transmission NOT success then
    Retry transmission
    if TIMEOUT then
        Place ID in transmission queue
    end if
end if
```

Server communication task

The Server communication task found on Arduino IR V1 have been re-implemented due to the lack of support for sending update messages and synchronization issues. The automatic re-transmission of message from the server require the syncing of a request number. The server send back an acknowledged message for any incoming message containing the request number. For solving the synchronization issue, the request number is fetched from all acknowledged message received from the server and used in all outgoing transmission from the robot. The new implementation is designed to bulk stream sensor updates, as well as, sending periodically out ping response to the server. The robot does therefore not implement re-sending of messages as it is not critical if an update message is lost. The ping response is used as an alive signal for checking if the server is responding. The new Server communication task offer a richer set of features and a more modular design. fig. 4.3 shows the program flow of the Server communication task.

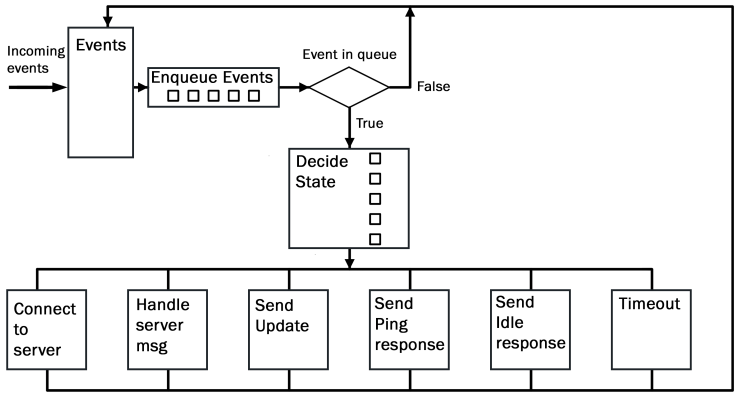


Figure 4.3: Block diagram illustrating the program flow of the Server communication task

The task is designed as a modular state machine where variables corresponding to a given state is queued into a FIFO queue based on a given event. The task decide the next state to run based on the first element in the queue. This allows for multiple events occurring at the same time and the task sequentially runs the states in queued order. The task uses a state machine with the following states:

Idle: Default state for the server communication task
Handle server message: Handles incoming messages
Send update: Sends update messages containing distance measurement obtained from the IR sensors merged with position estimate fetched from Node

1.
Connect to server: Transmit connection and handshake message every 2 seconds
Send ping response: Sends a ping response to the main server every 10 seconds for checking if the server is still operational.
Send idle response: Sends a idle response to the server, indicating that the robot is not busy.
Timeout: Clears a "connected-to-server" and signals the other task by sending a "disconnect-from-server" event.

Any generated events are checked in an event table and the corresponding state is queued into the state FIFO queue. The **Handle server message** state, polls a "message-received" flag that is set based incoming messages, handled in the nRF51 port layer asynchronously as described in section 2.3. A timeout counter is incremented when there is no new messages received and likewise reset every time the "message-received" flag is set. A timeout event is triggered if the timeout counter expires and the task disconnects from the server when the **Timeout** state is executed.

For sending updates, processed IR Measurement are buffered into an array which is used in combination with an index FIFO queue. If there is element in the index FIFO queue, the **Send update** is triggered, where the position estimate is combined with the IR measurement to form the update message. The events for the states **Connect to server**, **Send ping response** and **Send idle response** uses counters for triggering. A "connected-to-server" flag using the SR-latch described in section 2.3 is used along side the event counter for connecting to the server in the **Connect to server** state. The event counter for the **Send idle response** state is combined with a SR-Latch based "send-idle" flag. The flag is set based on a FreeRTOS event signaled by the Coordinator task. The **Send ping response** state is triggered periodically by the event counter.

Chapter 5

Improvements of the position estimate

From section 2.4 it was concluded that the errors in the position estimate greatly impacts the performance of the robot. This is most evident in the position controller, as it directly relies on accurate position estimate. Therefore, the implementation in chapter 6 and chapter 7 indirectly depend on improvements in the position estimate. The estimation method used in Arduino IR V1 was in some part empirical and makes it hard to reason about the cause of the inaccuracies. This motivates the investigation of the kinematic formulation and how the sensors values is processed based on physical parameters. Similar to Arduino IR V1, the position estimate in the new system is based on the sensor measurement from the encoders and the IMU. The first step to improve the position estimate was described in chapter 3 by replacing the original IMU and encoders. It is observed that the encoders have an accumulative effect on the error in the position estimate. This is a result of the encoders being dependent on the rotation of the wheels, which consequently introduce errors by wheel slippage. The accelerometer found on the IMU is independent of the wheels is utilized for combating the errors introduced by the encoders.

As described in section 2.3 and section 3.1, the gyroscope outputs the angular velocity, where numerical integration is used to calculate the heading of the robot. This makes the robot sensitive to noise and resulting in drifting of the heading. Arduino IR V1 made an effort of dealing with the drifting [19] by rejecting any value below a certain threshold. However, it had it limitation of not taking advantage of the statistical properties of the signal, as well as, using only a simple mean for smoothing. Kalman filtering is a method that do take advantage of these properties and can be used to give a more accurate measurement for signals corrupted by

noise. [10]. The method is briefly described in section .3.

5.1 Filtering of sensors data

A common form is additive noise and, within the software system on the robot, is first observed in the raw values extracted from the sensors. The focus within the improvements of the position estimate lays within the filtering of the raw sensor values. The filtering scheme applies a similar method found in Arduino IR V1. However, the statistical properties from the signals sampled from the gyroscope and accelerometer is utilized for better filtering, in addition to a tighter threshold in the signal rejection. In addition, Kalman filtering is used for the raw measurement from the gyroscope z-axis.

The sensor reading from the IMU can be classified as a Stochastic discrete random signal [11]. It is observed that the raw angular velocity and raw linear acceleration from the IMU described in section 3.1 is a constant added with noise when the robot is not moving.

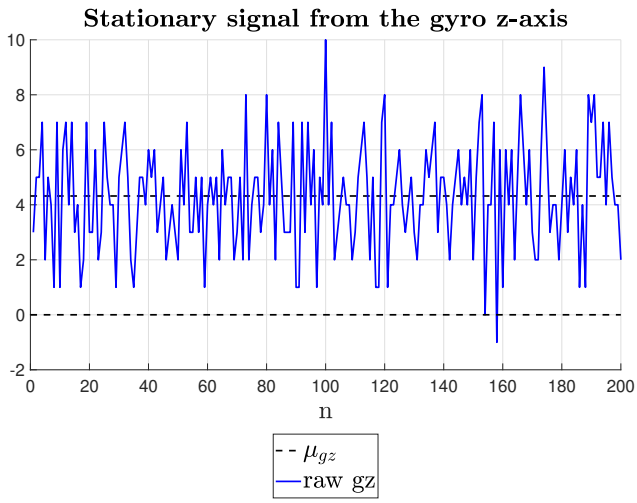


Figure 5.1: Recorded signal from the IMU gyro-z axis

.

fig. 5.1 shows the observed signal from the IMU gyro-z axis when the robot is stationary. Assuming the the raw signals from the gyroscope and the accelerometer takes the form of random signals with additive noise, it can be described by

eq. (5.1) when the robot is stationary and eq. (5.2) when the robot is moving.

$$r[n] = \mu_r + w \tag{5.1}$$
$$r[n] = X[n] + w \tag{5.2}$$

where the noise is assumed to be white Gaussian noise with zero mean and a variance σ^2. The signal in eq. (5.1) is also assumed to have a normal distribution centered around the mean μ_r, as well as, the properties of the noise is the same for eq. (5.1) and eq. (5.2). The robot is battery operated and the noise from the gyroscope and the accelerometer may increase over time. As a result, both μ_r and σ^2 is computed online on the robot using the sample mean defined in 5.3 [29] and the sample population variance defined in eq. (5.4) [29].

$$\mu = \frac{1}{n}\sum_{i=1}^{n} \tag{5.3}$$
$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2 \tag{5.4}$$

The static filtering applied for the accelerometer and gyroscope is carried out in three steps

1. Offset the signal to zero using the mean μ_r of the signal.
2. Filtering out high frequency noise using the digital low pass filter in eq. (2.1).
3. Reject any value below a threshold defined by $\pm\sigma$.

The Kalman filter in section .3 can be computational expensive as it relies on matrix operation if several states are to be estimated. A compromise is made, that favor system response, by only using Kalman filtering on the raw measurement obtained from the gyroscope z-axis. This is justified by noting that noise from the gyroscope is a mayor contributor to the error in the position estimate. As the Kalman filter is only used for the raw measurement from the gyroscope z-axis, all of the functions in section .3 can be implemented as scalar functions. The implemented scalar Kalman filter for the gyroscope z-axis is initialized by

$$h_{gz} = 1 + \Delta t \tag{5.5}$$
$$q = \sigma_{gz}^2 \tag{5.6}$$
$$r = \mu_{gz} \tag{5.7}$$

5.2 Kinematic model of the robot

The discrete kinematic equation used in Arduino IR V1 is altered to accompany the introduction of the accelerometer into the position estimate. The new system uses the definition of the coordinate frame R and W as described in section 2.3, as well as the position defined by eq. (2.5) and eq. (2.4). The frame R is set to be a inertial frame of reference to two new frames c_i and r_i. The robot fixed to the r_i frame moves in a straight relative to the x-axis in the R frame. The c_i frame stationary to the center of rotation rotates around the z-axis in the R frame.

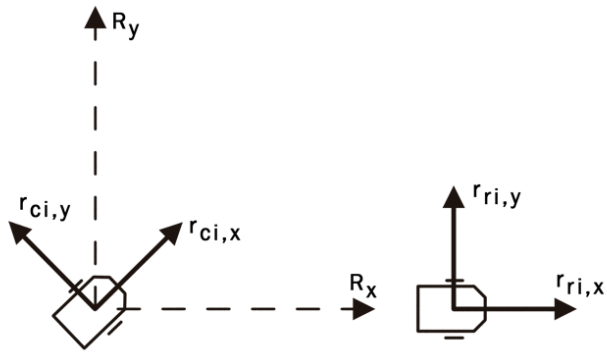


Figure 5.2: Illustration of the two coordinate frames c_i and l_i

Both frames have its origin in the center line of the wheel axis of the robot as illustrated in fig. 5.2. In the favor of embedded system design, the definition of the frames combined with the placement of the sensors allows the motion of the robot to be described with the linear and circular equation of motion in eq. (5.8) and eq. (5.9). The full kinematic equation for estimating the position of the robot is now:

$$\theta_{ci,k} = \omega_{gz,k}\Delta t \tag{5.8}$$

$$L_{ri,k} = v_{e,k}\Delta t + \frac{1}{2}a_{ax,k}\Delta t^2 \tag{5.9}$$

$$p_k^r = \begin{bmatrix} L_{ri,k}cos(\theta_{ci,k}) \\ L_{ri,k}sin(\theta_{ci,k}) \\ 0 \end{bmatrix}$$

$$p_k^w = p_{k-1}^w + R_r^w p_k^r$$

Δt is the sampling time, $\theta_{ci,k}$ is the current heading of the robot in the R frame, $\omega_{gz,k}$ is the sampled angular velocity from the IMU gyro z-axis, $v_{e,k}$ is the velocity

from the encoders and $a_{ax,k}$ is the linear acceleration from the accelerometer. The integration for obtaining the heading in the world frame uses the full trapezoid method found by bilinear transformation [11] defined as

$$\theta_{w,k} = \theta_{w,k-1} + \frac{\Delta t}{2}(\omega_{gz,k} + \omega_{gz,k-1}) \tag{5.10}$$

5.3 Position estimation task

Algorithm 5: Estimation task

```
Sample raw data from the IMU
switch (State)
case Calibrate:
    switch (Calculation mode)
    case Mean:
        Calculate μ_r using eq. (5.3)
        if μ_r ready then
            Go to Variance
        end if
    case Variance:
        Calculate σ_r^2 using eq. (5.3) ans μ_r
        if σ_r^2 ready then
            Go to state Operational
        end if
    end switch
case Operational:
    Sample C_r and C_l from the encoders
    Filter C_r and C_l using the Low pass eq. (2.1)
    Use static filtering on r_{a,x} and r_{g,x}
    Use Kalman filtering filtering on r_{g,z}
    Scale r_{a,x} using eq. (3.2) to obtain a_x
    Scale r_{g,z} using eq. (3.1) to obtain g_z
    Integrate g_z using eq. (5.10)
    Find position in frame R using eq. (5.9), eq. (5.8) and eq. (2.5)
    Find position in frame W using eq. (2.4)
end switch
```

The Position estimation samples the raw linear acceleration and the angular velocity from the IMU and the encoder speed ticks from the drivers described in section 3.2. The task is designed with two states **Calibrate** and **Operational**. The **Calibrate** state is run on system start up ,or when a system reset is performed, and calculates the sample mean and population variance for the raw measurement from the IMU. The position estimation is performed in the **Operational** state.

5.4 Evaluation of the position estimation

The gyroscope will be evaluated with respect to filtering and processing, as the it have the most effect on the accuracy of the position estimate. For testing the heading estimate with the new filtering scheme, the robot is rotated $\pm$ 90 degrees. The position of the robot is recorded and the raw values from the IMU is logged from the robot. The angle for each recorded sampling point is found using the $atan2$ in described in eq. (11). Lastly, a complete test, joining the described methods, for testing the position estimate is performed. This is carried out by pushing the robot in a random direction without causing the wheels to slip and record the position. The code for estimating the position is recreated within MATLAB and ran using the logged raw values, using the same types used in the robot. The parameters in table 4 is used for each the test.

Filtering and processing

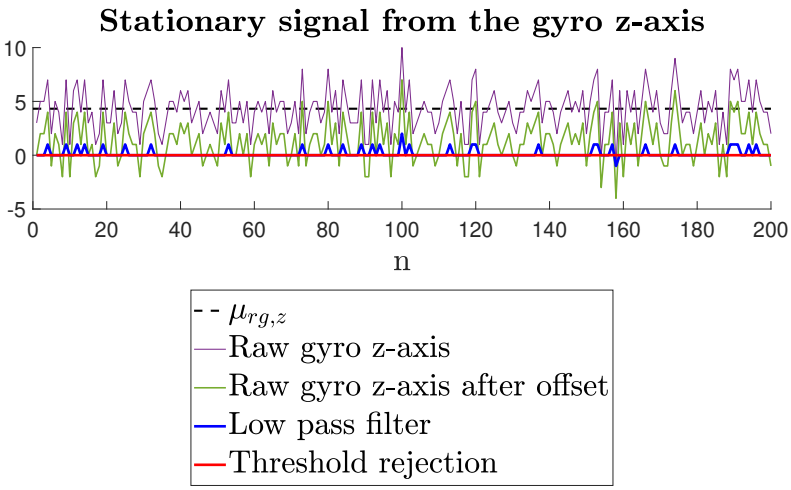


Figure 5.3: Static filtering of the stationary raw angular velocity from the IMU gyro z-axis

.

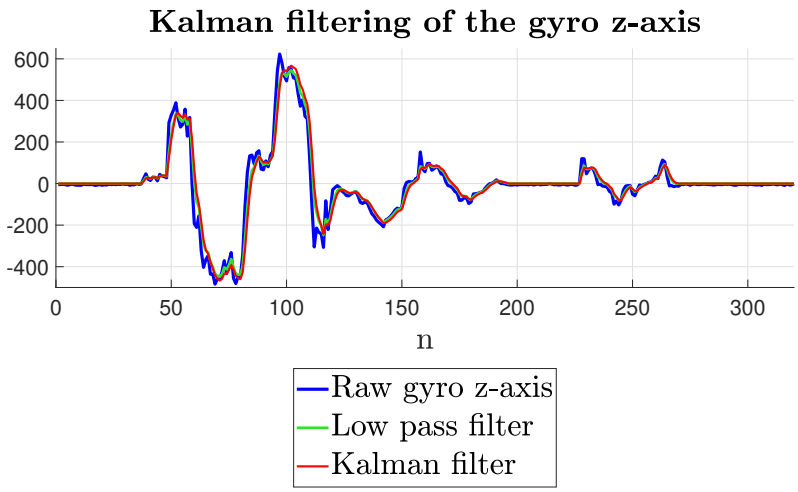


Figure 5.4: Kalman filter on the raw angular velocity from the IMU gyro z-axis

.

The result in fig. 5.3 shows how the static filtering of the raw signal from the gyroscope z-axis performs, where a sample mean $\hat{\mu} = 3$ and variance $\hat{\sigma}^2 = 4$ was used. Both $\hat{\mu}$ and $\hat{\sigma}$ may vary as they are calculated each time the robot is powered on. It is however observed that they tend to be around the values in table 5.1. By using the calculated sample mean and variance, the noisy raw signal is effectively removed as represented by the red line in fig. 5.3. Shifting the signal by $\hat{\mu}$ and utilizing low pass filtering in combination with utilizing the variance as bound in the threshold rejecting means that only raw values corresponding to angular velocity in the range of $[0.2, 0.5]$ $[dps]$ are lost. The trade-off is in favour of the static filtering, as the robot usually cannot rotate that slowly. Correspondingly, using the static filtering means that the heading obtained from eq. (5.10) does not dramatically shift and making the measurement from the gyroscope useless.

$\hat{\mu}$	$\sim$	$3-4$
$\hat{\sigma}^2$	$\sim$	4

Table 5.1: Nominal values for $\hat{\mu}$ and $\hat{\sigma}$

fig. 5.4 shows the effect of the Kalman filtering of the raw signal from the gyroscope z-axis, where the parameters q and r have been initialized with the calculated $\hat{\mu}$ and $\hat{\sigma}$. As shown by the red line, the combined filtering scheme of static filtering and Kalman filtering is successful in removing the additive noise in the raw signal, as well as, preserving the form of the raw signal. The

result also indicates that the use of $\hat{\mu}$ and $\hat{\sigma}$ in q and r is successful for providing automatic tuning of the Kalman filter.

Heading estimation

fig. 5.5 and fig. 5.6 shows the resulting estimated heading of the robot in the world frame compared to recorded heading. The estimating heading is compared with and without the Kalman filtering as shown in fig. 5.4. The red dotted line is the estimated angle with Kalman filtering, the green without and the blue is the real angle when rotating 90 and -90 degree respectively. The real angle is found by calculating the recorded position of the robot and using eq. (11). Using this method the real angle is in some form estimated. table 5.2 and table 5.3 shows the final angle, the final error, in addition to, the mean error between the recorded and estimated angle. The number of samples logged by the robot and the samples recorded from the Motion lab is not equal. To provide some numerical evaluation of the difference between the real and the estimated angle, a sample error is calculated by

$$e_{\theta,k} = |\theta_{r,k}| - |\theta_{e,k}|$$

where $\theta_{r,k}$ and $\theta_{e,k}$ are correspondingly sampled real and estimated points.

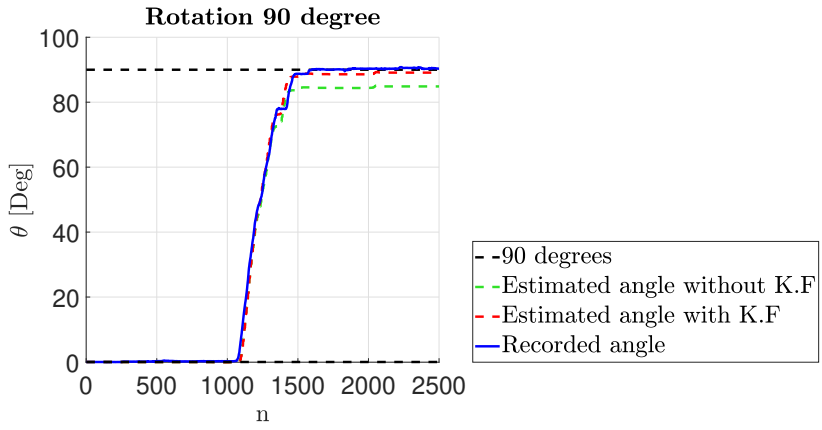


Figure 5.5: Estimated heading for rotation 90 degrees with and without Kalman filtering

.

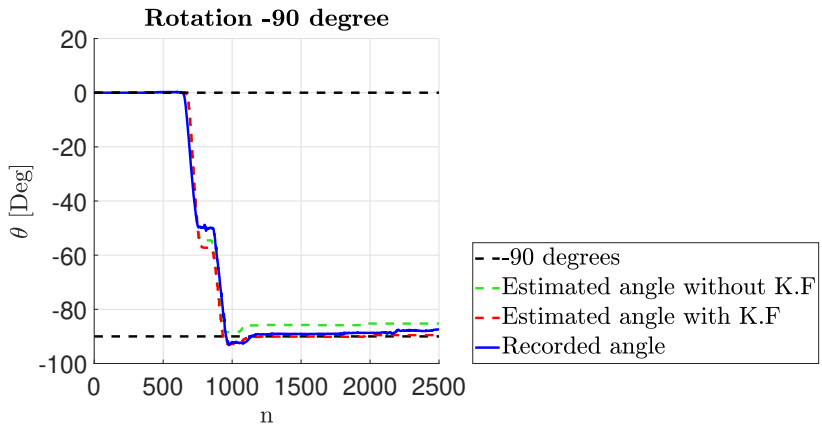


Figure 5.6: Estimated heading for rotation -90 degrees with and without Kalman filtering

.

Rotation 90 degree		
Recorded angle θ_r : 90.35	E. A with Kalman filtering θ_r : 89.10 e : 1.25 me 0.38	E. A without Kalman filtering θ_r : 84.86 e : 5.49 me : 2.15

Table 5.2: Statistics from the estimation of the 90 degree rotation

Rotation -90 degree		
Recorded angle θ_r : -89.10	E. A with Kalman filtering θ_r : -90.04 e : -0.944 me -0.75	E. A without Kalman filtering θ_r : -85.75 e : 3.34 me : 1.2607

Table 5.3: Statistics from the estimation of the 90 degree rotation

The result from fig. 5.5 and fig. 5.6 indicates good agreement when comparing the real and the estimated angle with applied Kalman filtering. It can be observed that the estimated angle without using Kalman filter is comparable with the Kalman filtered angle up to sample point 1300 and 900 for the positive and negative rotation respectively. However the intention with the Kalman filter is to make the

estimated angle more consistent a shown from fig. 5.4. Therefore, the mean error 0.38 -0.75 from table 5.2 and table 5.3 for the positive and negative estimated angle with Kalman filtering gives an indication of a more accurate measurement. The final error of 5.4 and 3.3 when not using the Kalman filtering for the positive and negative estimated angle is consistent with final values observed from logging when rotating the robot. The mean error of 2.6 and 1.2 is also fit the observed offset when rotating the robot back to zero. Likewise, when using the Kalman filtering for heading estimation, an error in the range of ± 1 degree has been observed, indicating that the final values of 1.2 and -0.94 are reasonable.

An important observation to be made when using the Kalman filtering is the negative sign in the mean error for rotation -90 degree. This suggest that the estimated heading using the Kalman filter will be ahead when rotation in the negative direction and lag behind when rotating in the positive direction. The result from fig. 5.5 and fig. 5.6 also gives an agreement of this observation. As a consequence, rotating the robot in one direction over a extended period of time will eventually result in deviation between the real and the estimated heading. Regardless, under normal operation the robot will not drive exclusively in one direction. It has been observed that the drift error tend to be minimal when rotating the robot equally in a positive and negative direction.

Position estimation

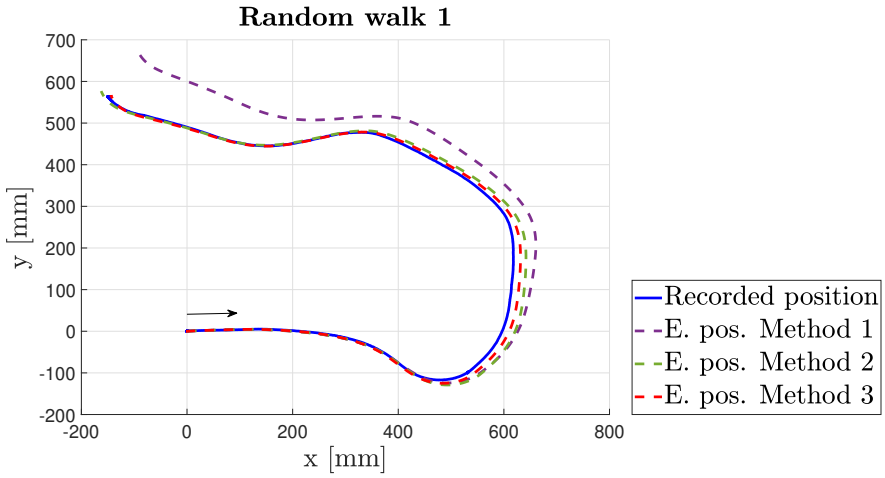


Figure 5.7: Comparison between the estimated and recorded position from random walk 1

.

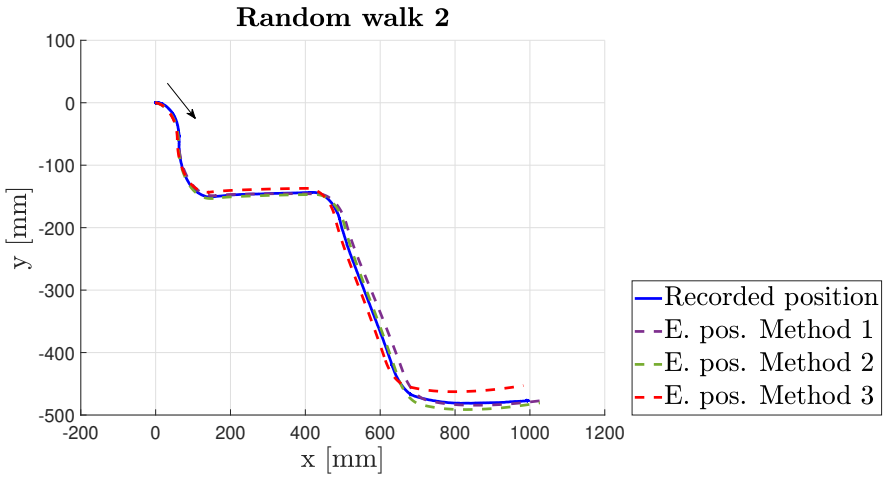


Figure 5.8: Comparison between the estimated and recorded position from random walk 2

.

fig. 5.7 and fig. 5.8 shows the estimated position of the robot in the world frame and the recorded position. The estimation relies on the filtering scheme in section 5.1 and the kinematic equation eq. (2.5) in section 5.2. The recorded position is compared against three estimation methods. Method one is the estimated position without Kalman filtering and accelerometer measurement. The second is the estimated position with Kalman filtering and without accelerometer measurement. The third method uses both Kalman filtering and the measurement from the accelerometer. For obtaining the final error and the mean error for each estimation method, a similar calculating as the heading estimation was done to compare the estimated against the recorded position. The distance from the origin was also calculated for each sampled point.

Random walk 1 [mm]				
	$r_x: -149.55$	$r_y: 562.27$		
Method 1	$e_x: 53.46$	$e_y: -121.97$	$me_x: -3.79$	$me_y: -44.63$
Method 2	$e_x: -13.03$	$e_y: -14.55$	$me_x: -12.51$	$me_y: -6.09$
Method 3	$e_x: 5.89$	$e_y: -2.80$	$me_x: -3.13$	$me_y: -2.03$
	$d_r: 581.82$			
Method 1	$d_2: 690.96$		$e_d - 109.13$	$me_2 - 53.98$
Method 2	$d_3: 599.30$		$e_d - 17.47$	$me_3 - 13.41$
Method 3	$d_1: 583.05$		$e_d - 1.22$	$me_1 - 4.75$

Table 5.4: Error statistics for random walk 1

Random walk 2 [mm]				
	$r_x: 991.90$	$r_y: -476.53$		
Method 1	$e_x: -51.24$	$e_y: 0.51$	$me_x: -22.88$	$me_y: -1.0$
Method 2	$e_x: -33.73$	$e_y: -4.10$	$me_x: -11.23$	$me_y: -4.41$
Method 3	$e_x: 10.18$	$e_y: 23.55$	$me_x: 15.86$	$me_y: 14.28$
	$d_r: 1100.43$			
Method 1	$d_2: 1146.62$		$e_d - 46.19$	$me_2 - 20.81$
Method 2	$d_3: 1132.67$		$e_d - 32.23$	$me_3 - 12.401$
Method 3	$d_1: 1081.18$		e_d 19.25	me_1 20.44

Table 5.5: Error statistics for random walk 2

As seen in fig. 5.7, the deviation for method one is considerable, whereas both method two and three are close to matching the recorded position. The result in table 5.4 indicates that method three produces an average error in the x-and y-direction in the range of 5 millimeters, whereas method two have an average error in the range of 15 millimeters for random walk one. The result from also fig. 5.7 gives a strong indication for the importance of the Kalman filtering. On the contrary, all of the methods, within a reasonable degree, matches the recorded position in fig. 5.8. A reason for this, is that the path in fig. 5.7 have considerable more arcing, meaning the heading of the robot is constantly shifting. In contrast, the path from random walk two in fig. 5.8 resembles more of a series of connected lines. Still, the result from table 5.5 indicates, that method one have an average error in the x-and y-direction in the range of 20 millimeters. table 5.5 also reviles that

method two, which do not utilize the accelerometer, performs better than method three. However the calculated mean error and final error in the x-and y-direction for method two is not as consistent as method three. Aspects to be aware of is that the accelerometer is sensible to vibrations, as well as a less smooth motion was needed to form the path in fig. 5.8 comparing to the path in fig. 5.7. This gives an indication that jerking in the motion of the robot seems to negative affect the position estimate when using the accelerometer. However, this seems to only impact the final position for method three in fig. 5.8. Isolated, the paths from fig. 5.7 and fig. 5.8 produces somewhat conflicting results. However, combining the behavior observed from random walk one and two with the result in table 5.4 and table 5.5 indicates that the average error in the x -and y-axis for both cases is within the range of 15 millimeters.

Even thought the accelerometer is sensible to vibration and jerking in the motion of the robot it is still beneficial to use it in the position estimate. The result from the heading estimation reviled that rotating the robot in both positive and negative direction tend to minimize the drift error. Similar, the accelerometer possesses the properties for combating the accumulated error produced by the encoders. As observed from both table 5.4 and table 5.5 the error for method two is constantly negative. This means that using method two which, relying only on the encoders for the translation motion, would result in a constant shift in the estimated compared to the real position. In contrast, the result for method three indicates that using the accelerometer would produce an error such that the estimated position would tend to stay within a region of the real position.

Chapter 6

Control of the robot

The assessment of Arduino IR V1 in section 2.4 found that using PID controllers only gives adequate control for the motion of the robot. As concluded in section 2.4, the main issue is the oscillatory behaviour and large overshoot experienced when the robot is maneuvering, along with disturbance produced by the motors used in Arduino IR V1. This is partly resolved by replacing them as described in section 3.1. Still, the response for each motor may not be equal and a speed controller for each motor is required to decrease the difference. The "black box" design of Arduino IR V1 does not take in to account the dynamics of the robot. Applying the linear and angular velocity into a state feedback control law on the form of eq. (6.1) [9] can better suit the MIMO system for controlling the robot to a target coordinate.

$$\mathbf{u} = -\mathbf{K}\mathbf{x} \tag{6.1}$$

However, this require a linear time invariant description of the system on the form

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} - \mathbf{B}\mathbf{u} \tag{6.2}$$

The state feed back can be combined with reference feed forward to form the following control law in eq. (6.3), using the definition in eq. (6.4) for $\mathbf{K}_r$ [23].

$$\mathbf{u} = \mathbf{K}_r\mathbf{r} - \mathbf{K}\mathbf{x} \tag{6.3}$$

$$\mathbf{K}_r = [(\mathbf{B}\mathbf{K} - \mathbf{A})^{-1}\mathbf{B}]^{-1} \tag{6.4}$$

Granted that a reasonable description exist, as well as being time invariant and controllable, any state of the system can be brought to another in finite time by applying a appropriate control output u [9]. Another advantage of applying state feed back is that optimal control in the form of LQR *Linear quadratic regulator* can be used, as eq. (6.1) is the solution for minimizing the quadratic cost function

defined in eq. (6.5) [9] [2].

$$J = \int_0^\infty \mathbf{x}^T(t)\mathbf{Q}\mathbf{x}(t) + \mathbf{u}^T(t)R\mathbf{u}(t) \tag{6.5}$$

where $\mathbf{Q}$ and $\mathbf{R}$ are positive definite matrices and

$$\mathbf{K} = \mathbf{R}^{-1}\mathbf{B}^T\mathbf{S} \tag{6.6}$$

is the solution to the Algebraic Riccati Equation in eq. (6.7) [9].

$$\mathbf{A}^T\mathbf{S} + \mathbf{S}\mathbf{A} - \mathbf{S}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{S} + \mathbf{Q} = 0 \tag{6.7}$$

The proposed solution is to obtain a linear dynamic model of the robot and use it to design a reference feed forward state feedback control law on the form of eq. (6.3). The controller is designed as a LQR controller where the gain matrix $\mathbf{K}$ that minimize eq. (6.5) along with eq. (6.4) is found using `MATLAB`. The calculated values of $\mathbf{K}$ and $\mathbf{K}_r$ is transferred to the Atmega2560 controller and an explicit control law of eq. (6.3) is implemented. The complete controller is structured as a cascaded system, where the LQR controller is responsible for guiding the robot to a target coordinate. The outputs from the controller is used as reference in a speed controller, which is responsible for maintaining a given speed for each wheel. The speed controller make use of the discrete PID implementation in eq. (2.3) described in section 2.3.

The chapter can be structured in the following way

- section 6.1 Dynamic model of the robot
- section 6.2 Optimal control using LQR
- section 6.3 System response
- section 6.4 Square test of the new controller

6.1 Dynamic model of the robot

Using LQR require a linear model for describing the dynamic of the robot. The kinematic equation in eq. (2.5) is clearly not linear and cannot be used without linearization. Moreover, the dynamics is not exclusively dependent on being described in the world frame. Justified by the fact that the robot is designed to move around on a flat surface where no external forces, except friction, is assumed to apply in the horizontal direction. Furthermore, the controller is in its basic form a velocity controller and the dynamics can be described with respect to the linear and angular velocity of the robot.

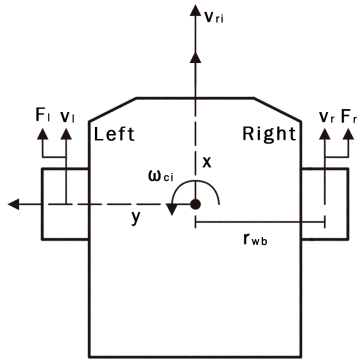


Figure 6.1: Dynamic model of the robot

The behaviour of the robot is described using two separate coordinate frames, a transnational and rotational similar to the frames c_i and r_i in section 5.2. The Euler-Lagrange equation described in section .4 is used to formulate the dynamic model of the robot. The total kinetic energy of the robot with the general coordinates $\mathbf{q} = [d_{ri}, \theta_{ci}]$ is formulated in eq. (6.8), where m_r is the mass and J_r is the moment of inertia of the robot.

$$\mathcal{K} = \frac{1}{2} m_r \dot{d}_{ri}^2 + \frac{1}{2} J_r \dot{\theta}_{ci}^2 \tag{6.8}$$

Using the Euler-Lagrange equation in eq. (9) for $[d_{ri}\theta_{ci}]$ and eq. (6.8) result in

$$m_r \ddot{d}_{ri} = F_{ri}$$
$$J_r \ddot{\theta}_{ci} = T_{ci}$$

As shown in fig. 6.1, the force F_l and F_r from the left and right wheel is the applied forces of the robot and r_{wb} is the wheel base radius. The force F_{ri} can be expressed as the average between the horizontal force from the left and right wheel. The torque T_{ci} is expressed as $T_{ci} = r_{wb}(F_r - F_l)$, which result in the dynamic equations in eq. (6.9) and eq. (6.10).

$$m_r \ddot{d}_{ri} = \frac{1}{2}(F_r + F_l) \tag{6.9}$$
$$J_r \ddot{\theta}_{ci} = r_{wb}(F_r - F_l) \tag{6.10}$$

As the robot is categorized as a differential drive mobile robot, the two motors with wheels are the main actuators of the robot. The dynamics of each wheel with

motor can be modeled by eq. (6.11) [22], where T_u is the input torque, T_L is the load torque acting on the motor and J_m is the moment of inertia of the wheel.

$$J_m\dot{\omega}_{m,r,l} = T_{u,r,l} - T_L \tag{6.11}$$
$$T_L = T_c + B\omega_m \tag{6.12}$$
$$T_{u,l,r} \geq T_c \tag{6.13}$$
$$T_c \leq T_{u,l,r} \leq r_w F_c \tag{6.14}$$

The load torque T_L is viewed as a friction torque and simplified model of T_L is stated in eq. (6.12), where T_c is the Coulomb friction and the dampening B is a coefficient of the viscous friction [22]. The dampening is related to the internal losses in the motor. The static friction can be view as the minimum torque required to make the wheels rotate when in contact with the surface, constraining the input torque to eq. (6.13).

The wheels need to satisfy the constraint of rolling without slipping, such that $v_m = r_w\omega_m$ can be used [9]. The total constraint of the input torque is expressed in eq. (6.14), where F_c is the static coulomb friction given by $F_c = \mu_f m_R g$, for a friction coefficient μ_f [22]. Another aspect is that T_u have generally not a linear response. However the dynamic equations are to be applied to the controller and implemented into the software system of the robot, where the torque is controlled by a PWM signal as described in section 2.2 and section 3.1. This result in the freedom of simply finding the necessary mapping such that the region $[T_{u,min}T_{u,max}]$ is a linear operation region and that any $T_u \neq 0$ result in a movement of the robot, without slipping of the wheels, assuming constant μ_f. Applying this, the input torque can be described as

$$\hat{T}_u = (T_u - T_c) + T_{u,min} \tag{6.15}$$

This result in the following dynamic model for the left and right motor with wheels

$$\dot{\omega}_{ml} = \frac{1}{J_w}\left(\hat{T}_{ul} - B\omega_l\right) \tag{6.16}$$
$$\dot{\omega}_{mr} = \frac{1}{J_w}\left(\hat{T}_{ur} - B\omega_r\right) \tag{6.17}$$
$$\dot{\omega}_{ml} = \frac{r_w}{J_w}F_l \tag{6.18}$$
$$\dot{\omega}_{mr} = \frac{r_w}{J_w}F_r \tag{6.19}$$

Using eq. (6.19) and eq. (6.18) in eq. (6.9) and eq. (6.10), the dynamical model of

the robot can be expressed as

$$\ddot{d}_{ri} = \frac{1}{2}\frac{1}{m_r}\left(F_d - B\dot{d}_{ri}\right) \tag{6.20}$$

$$\ddot{\theta}_{ci} = \frac{r_{wb}}{J_r}\left(F_\theta - B\dot{\theta}_{ci}\right) \tag{6.21}$$

where the moment of inertia of the robot is modelled as a solid rod in eq. (6.22)

$$J_r = \frac{1}{12} m_r w b^2 \tag{6.22}$$

and $\dot{d}_{ri}$, $\dot{\theta}_{ci}$, F_d and F_{theta} found from the relationships

$$\begin{aligned}
\dot{d}_{ri} &= \frac{1}{2}(\frac{v_r + v_l}{r_w}) = v_e \\
\dot{\theta}_{ci} &= r_{wb}(\frac{v_r - v_l}{r_w}) = \omega_{g,z} \\
F_d &= T_{ur} + T_{ul} \\
F_\theta &= T_{ur} - T_{ul}
\end{aligned}$$

The dynamics of the robot in eq. (6.20) and eq. (6.21) is described as two linear second order differential equations. By reformulating $\dot{d}_{ri}$ and $\dot{\theta}_{ci}$ as v_{ri} and ω_{ci}, a continuous state space equation on the form

$$\begin{aligned}
\dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \\
\mathbf{y} &= \mathbf{C}\mathbf{x}
\end{aligned}$$

with the states $\mathbf{x} = [v_{ri}\ \omega_{ci}]^T$ and $\dot{\mathbf{x}} = [\dot{v}_{ri}\ \dot{\omega}_{ci}]$ can be formulated as

$$\begin{bmatrix} \dot{v}_{ri} \\ \dot{\omega}_{ci} \end{bmatrix} = \begin{bmatrix} -\frac{B}{2m_r} & 0 \\ 0 & -\frac{Br_{wb}}{2J_r} \end{bmatrix} \begin{bmatrix} v_{ri} \\ \omega_{ci} \end{bmatrix} + \begin{bmatrix} \frac{1}{2m_r} & 0 \\ 0 & \frac{r_{wb}}{J_r} \end{bmatrix} \begin{bmatrix} u_{ri} \\ u_{ci} \end{bmatrix} \tag{6.23}$$

and

$$\begin{bmatrix} y_{ri} \\ y_{ci} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_{ri} \\ \omega_{ci} \end{bmatrix} \tag{6.24}$$

where $\mathbf{u} = [u_{ri}\ u_{ci}] = [F_d\ F_\theta]$ and v_{ri} and ω_{ci} is provided by the estimated linear and angular velocity from the estimation task. Utilizing the controllability matrix defined in eq. (10) [5], the system is found to be controllable where $rank(\mathcal{C}) = 2$.

6.2 Optimal control using LQR

The complete controller is structured as a cascaded system. The LQR controller is responsible for guiding the robot to a target coordinate and uses the state feedback with reference feed forward described in eq. (6.3) with the model in eq. (6.23), where $\mathbf{K}$ and $\mathbf{K}_r$ is found using eq. (6.5) and eq. (6.4). The calculated output from the LQR controller is used as an reference to the a speed controller, responsible for maintaining a given wheel speed. The velocity of each wheel is controlled by two separate discrete PID controller described by eq. (2.3) in section 2.3. fig. 6.2 shows a block diagram of the complete controller.

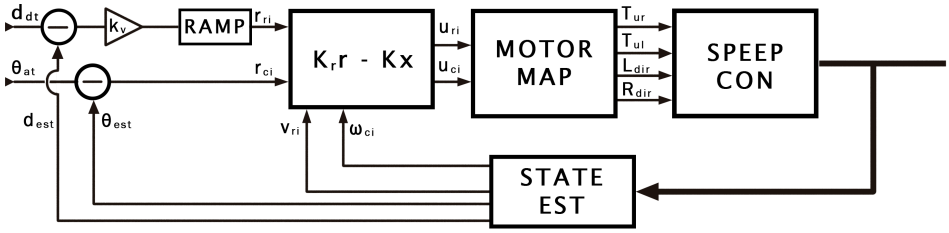


Figure 6.2: Block diagram of the complete controller with the LQR reference feed forward state feedback controller

To run the compact expression in eq. (6.3) and solving eq. (6.7) within the system on the Atmega2560 controller is not feasible. Taking advantage of the time invariant properties of $\mathbf{A}$ and $\mathbf{B}$ in eq. (6.23), the matrix $\mathbf{K}_r$ and $\mathbf{K}$ is computed offline using `MATLAB`. The matrix $\mathbf{K}$ and found using the built in function `lqrd` [15] and $\mathbf{K}_r$ is found using eq. (6.4). The implemented LQR controller on the Atmega2560 runs the explicit control law in eq. (6.25) and eq. (6.26).

$$u_{d,k} = r_{11} r_{v_{ri},k} - k_{11} v_{e,k} \tag{6.25}$$

$$u_{\theta,k} = r_{22} r_{\omega_{ci},k} - k_{22} \omega_{g,z,k} \tag{6.26}$$

Similar to Arduino IR V1, the LQR controller takes in a target set-point coordinate $[x_s p \ y_s p]$ and the goal is to $d_{dt} \rightarrow 0$ and $\theta_{da} \rightarrow 0$ for the distance and heading defined in eq. (2.15) and eq. (2.16). Arduino IR V1 had issues with the computed θ_{da} [19]. This is solved by using eq. (1) in combination with eq. (2). The LQR controller guides the robot to the target coordinate in a sequentially manner using two logical sates **Align to target** and **Move to target**. The robot is first aligned to the target coordinate based on θ_{da} using only the angular state feedback in eq. (6.26). When the robot is within a angle threshold $\delta_{thrs,\theta}$, the robot stops and the movement of the robot is controlled in the **Move to target** state using both eq. (6.25) and eq. (6.26). The target is considered reached when the robot is within

a region threshold $\delta_{thrs,d}$. The reference state $\mathbf{r} = [r_{v_{ri}} r_{\omega_{ci}}]$ in the LQR controller utilize the values of d_{dt} and θ_{da}. As the motion of the robot is controlled sequentially, only $r_{\omega_{ci}}$ is used in the **Align to target** state. In this state, the reference $r_{\omega_{ci}}$ is ramped up to a value proportional to θ_{da} defined in eq. (6.27). $r_{\omega_{ci}}$ uses θ_{da} directly in the **Move to target** state.

$$r_{\omega_{ci},k} = \begin{cases} \delta_{ci,k} & |\delta_{ci,k-1}| > |r_{\omega_{ci},k-1}| \\ r_{\omega_{ci},k-1} - s_{\text{ramp}} & \text{sign}(\delta_{ci,k-1}) < 0 \\ r_{\omega_{ci},k-1} + s_{\text{ramp}} & \text{sign}(\delta_{ci,k-1}) > 0 \end{cases} \tag{6.27}$$

In similar fashion, a value proportional to d_{dt} is used for $r_{v_{ri}}$ in the **Move to target** state. To make the robot accelerate smoothly and enforce the non-slip condition, $r_{v_{ri}}$ is ramped up to a value proportional to $\delta_{ri,k}$, defined in eq. (6.29), by a step size s_{ramp} defined by eq. (6.28), where K_v is an appropriate scaling factor.

$$r_{v_{ri},k} = \begin{cases} r_{v_{ri},k-1} + s_{\text{ramp}} & r_{v_{ri},k-1} < \delta_{ri,k} \\ \delta_{ri,k} & r_{v_{ri},k} \geq \delta_{ri,k} \end{cases} \tag{6.28}$$

$$\delta_{ri,k} = \frac{d_{dt,k}}{K_v} \tag{6.29}$$

The control outputs $[u_{ri} \;\; u_{ci}]$ in eq. (6.25) and eq. (6.26) are feed into the motor mapper where the references $[T_{ref,l} \;\; T_{ref,l}]$ as well as the motor directions $[R_{dir} \;\; L_{dir}]$ are found. The outputs $[u_{ri} u_{ci}]$ is transformed to $[T_{ref,l} T_{ref,r}]$ using the mapping method described in table 2.3. To gain better response when the robot is turning and adapting the new motors described in section 3.1, the intermediate values T_{inc} and T_{rec} is now defined as

$$\begin{bmatrix} T_{inc} \\ T_{rec} \end{bmatrix} = \begin{cases} \begin{bmatrix} u_x - I_{TT} \\ u_x - I_{TT} \end{bmatrix} & u_x > I_{TT} \\ \begin{bmatrix} sat(u_y + 2u_x) \\ sat(u_y - 2u_x) \end{bmatrix} & u_x < I_{T1} \\ \begin{bmatrix} sat(u_y + 2u_x) \\ 0 \end{bmatrix} & I_{TT} > u_x > I_{T1} \end{cases} \tag{6.30}$$

The speed controller is responsible for maintaining a reference speed for each wheel, where a feed forward PID controller used to maintain a reference wheel tick speed based on a reference input T_{ref}. fig. 6.3 shows a block diagram of

the speed controller. The speed controller uses eq. (6.31) for maintaining a given left and right wheel speed based on the tick speed from the encoders, utilizing the implemented discrete PID controller in eq. (2.3) under section 2.3

$$T_{u,r,l} = T_{ref,r,l} \pm u_{\mathrm{PID},r,l} \tag{6.31}$$

The reference input $T_{ref,l,r}$ is both used as a baseline for $T_{u,l,r}$ and as a reference in the PI controller, which add or subtract the baseline value using the error in eq. (6.32).

$$e_{r,l} = \frac{T_{ref,l,r}}{10} - \mathrm{LinMap}(v_{m,l,r}, [v_{m,min}\ v_{m,max}], [0\ \ 100]) \tag{6.32}$$

where the measured tick speed is mapped to a range $[0 \quad 1000]$ using the LinMap function in eq. (2.2).

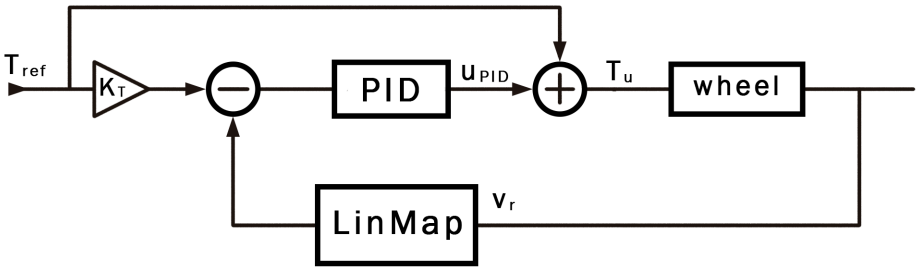


Figure 6.3: Block diagram of the PID controller used to maintain a reference tick spped

LQR control task

The task is designed with several states defined as:

- **Check for new events**
- **Compute control output**
- **Set stop**
- **Send output**

As illustrated from fig. 6.4, the task starts at the **Check for new events** state, where it checks if the control signals; start, stop and reset have been set, in addition to, received target set-point coordinates and position updates. The control signal set internally latch variables. The control signals are in the form of FreeRTOS events that are generated by the Coordinator task at Node 2 and transferred over the node

communication link. Both target set-point coordinates and position updates are in the form of FreeRTOS queues. The position updates are sent from the Position Estimator task.

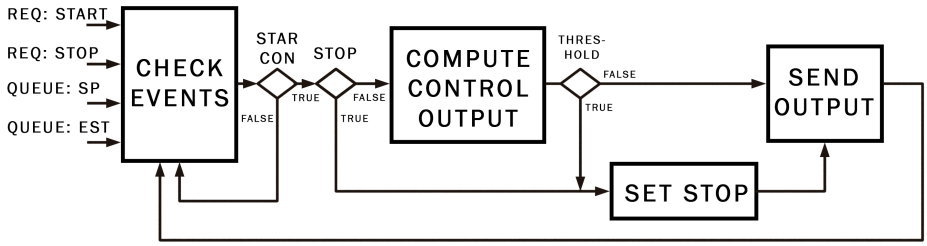


Figure 6.4: Flow chart for the LQR controller task

A start condition is used for allowing the task to transition to the **Compute control output** state. The target set point coordinate is latched when it is received and reset when the robot reaches the target coordinate. The start condition requires the start-latch to be set in addition to the target set point coordinate latch and a new position update. The task goes to the **Compute control output** state if the start condition is meet and no stop signal have been issued. Within the **Compute control output** state, the two states **Align to target** and **Move to target** are used along side eq. (2.15) and eq. (2.16) for computing d_{dt} and θ_{at} from (x_{sp}, y_{sp}).

For both internal cases, the task transition to the **Send output** if the threshold condition is not meet. If the condition is meet, then the task will first go the the **Set stop** state, where both u_{ri} and u_{ci} is zeroed and the "start-control" latch is reset along side the "target set-point" latch. The task immediately returns to the **Send output** state, where the output from **Compute control output** or **Set stop** is transferred to the Speed controller task via a FreeRTOS queue. The task will return to **Check for new events** after the output have been sent and the cycle the procedure.

The internal states **Align to target** and **Move to target** in **Compute control output** allows the task to control the alignment and movement separately. However in normal operation, the task execute the **Align to target** first followed by **Move to target**. As described previously, only the reference r_{ci} and the output u_{ci} is used in the **Align to target** state. The state uses the threshold condition in eq. (6.33).

$$|\theta_{da}| < |\frac{\theta_{at}}{5}| \tag{6.33}$$

The state **Move to target** uses the full reference feed forward state feedback con-

trol law in eq. (6.25) and eq. (6.26). The heading reference r_{ci} uses θ_{da} directly and the distance reference r_{ri} is computed by eq. (6.28). The threshold condition in **Move to target** make use of the position update, where (x_r, y_r) has to be within a region of (x_{sp}, y_{sp}) defined by the threshold δ_{thrs} and

$$(x_{sp} - \delta_{thrs}) \leq x_r \leq (x_{sp} + \delta_{thrs}) \wedge (y_{sp} - \delta_{thrs}) y_r \leq (y_{sp} + \delta_{thrs})$$

is computed by eq. (6.27) and the heading gain output is computed by the control law in eq. (6.26).

Wheel speed control task

The task is simple in design. The Speed controller receive the output gain $\mathbf{u}$ from the LQR controller and applies the motor mapper in eq. (6.30) and table 2.3 to obtain T_{Right} T_{Left} from $\mathbf{u}$. The task stops the motors if the reference is zero, else it maintains the given reference speed using eq. (6.31) as shown in fig. 6.3. The motors are also stopped if the task doesn't receive any new reference from the LQR controller task or if the encoders measure zero over a given period of time. Listing in algorithm 6 shows an simplified representation of the task.

Algorithm 6: Speed controller task

```
Uses motor mapper on u to find T_{Left,Right} and Dir_{Left,Right}
if Stop Or Timeout then
    Stop motors
else
    Obtain C_r and C_r by sampling the encoders
    Map C_{l,r} to T_{ref,l,r}
    Apply eq. (6.31) on the left and right motor.
    if C_r Or C_r = 0 then
        increment Timeout counter
    end if
end if
```

6.3 System response

Speed Controller

The reference input torques $[10, 100, 200, ..., 900, 1000]$ is used for testing the open and closed loop response of the Speed controller. The tick speed from each encoder is logged from the robot and the respective reference input torques is applied long enough for the wheels to reach a steady state. The left and right PI motor controller uses the K_P and K_I gains in table 6.1.

Right	$K_p = 2.7$	$K_I = 1.2$
Left	$K_p = 2.6$	$K_I = 1.1$

Table 6.1: Gains for the left and right PI motor controller

Result

fig. 6.5 and fig. 6.6 shows the result of the open and closed loop response. Each point correspond to a average tick speed calculated in the steady state region for each encoder. The input torque correspond to a mapping of the PWM signal that is applied on the motor controller card. fig. 6.5 shows that the mapped torque produces a response for the left and right motor that is linear up to a value of 800 and that there is significant difference in the angular tick speed. This confirms that the input torque range $[T_{u,min} \quad T_{u,max}]$ gives a operation region that is close to being linear. This is a vital property, as the feed forward PI controller in eq. (6.31) and the mapping in eq. (6.32) assume linear response.

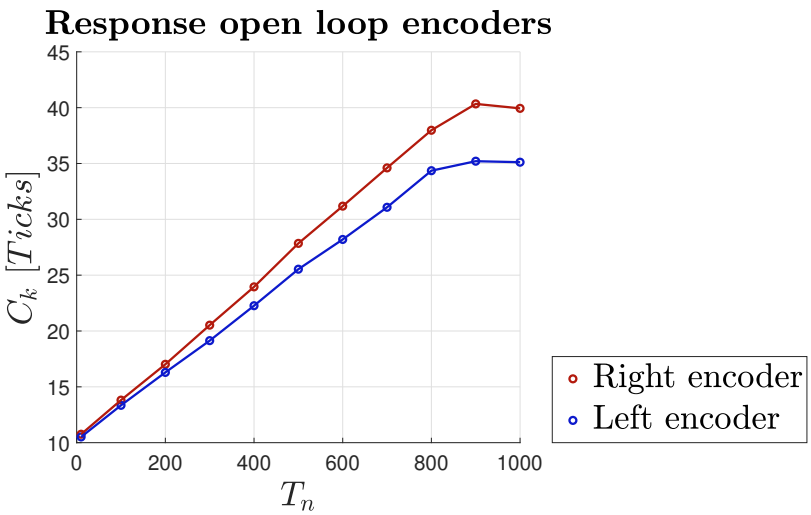


Figure 6.5: Open loop response for the input torque and average tick speed for the left and right encoders

.

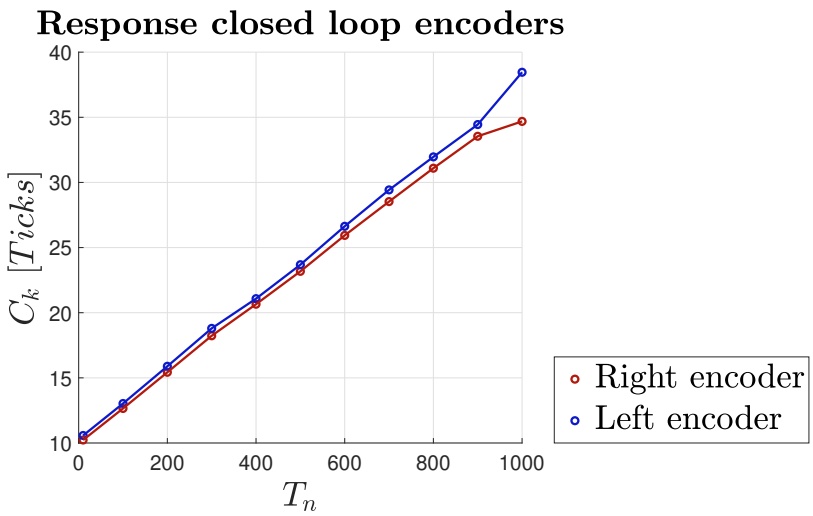


Figure 6.6: Closed loop response for the input torque and average tick speed for the left and right encoders

.

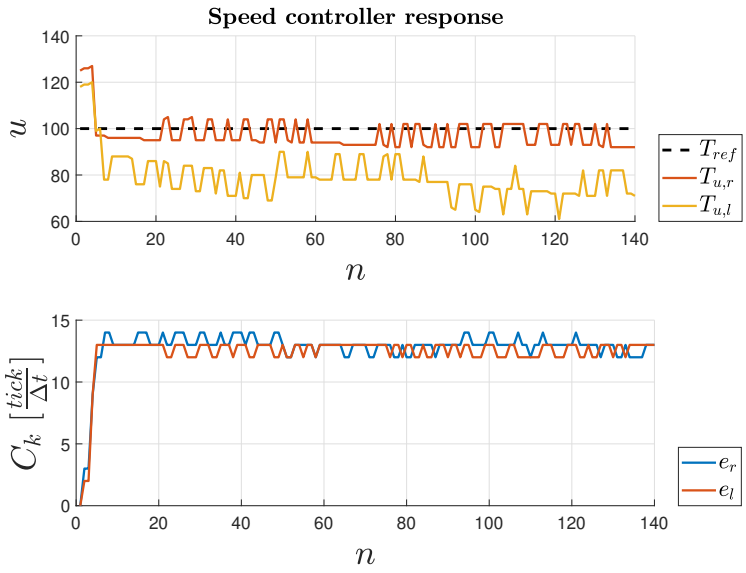


Figure 6.7: Response of the speed controller with $T_{ref} = 100$

.

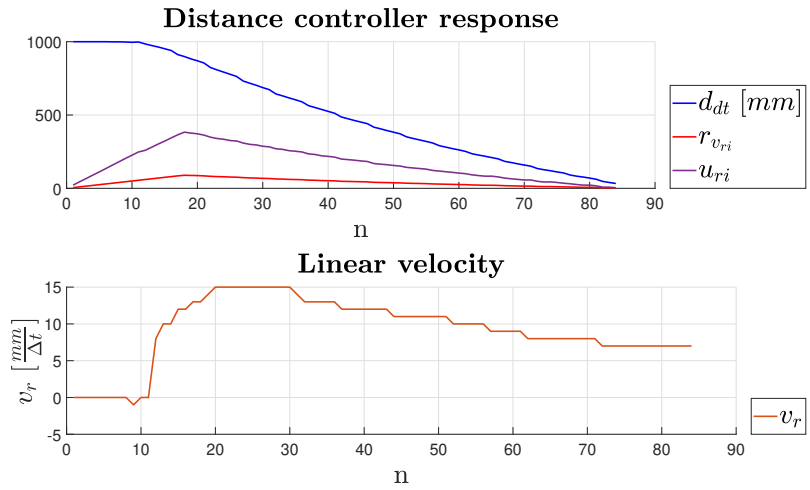
fig. 6.6 shows that the implemented PI speed controller produces a satisfactory output gain for matching the speed of the left and right wheel. fig. 6.7 shows the steady state response for a reference input torque of 100. As observed from fig. 6.7, the applied output gain $T_{u,l}$ and $T_{u,r}$ result in an equally response for the left and right wheel. It also produces a relative fast response, where the both the left and right wheel reaches a steady state within 5 iterations. The speed controller runs on a period of 25 ms and means that the controller is able to bring each wheel to a equal speed within 125 ms. fig. 6.7 also shows small ripples in the steady state region. The ripples are the reason for using a PI and not a full PID controller. In addition, the ripples will produce a slight offset between the left and right wheel and the robot will still experience a small disturbance when applying a constant torque. This shows that heading correction is necessary. Nevertheless, the speed controller is not the main control mechanism in the system and provide adequate adjustment.

LQR controller

The LQR controller is the master controller in the cascaded system. The speed controller have shown to give a reasonable response and is able to match the speed of each wheel. The heading controller is the most important part of the system. Therefore, the LQR controller is tested in an extreme situation where the controller only use the **Move to target** and a target set-point coordinate of $(0, 1000)$ mm is issued to the robot. The heading controller is also tested in the normal condition, where the robot is rotated 180 degree by issuing a target set-point coordinate of $(-500, 0)$. For testing the system response, the gain for $\mathbf{K}$ and $\mathbf{K}_r$ was found using the function `lqrd` [15] and `lqr` [14] in `MATLAB` based on eq. (6.23) and the parameters in table 5.

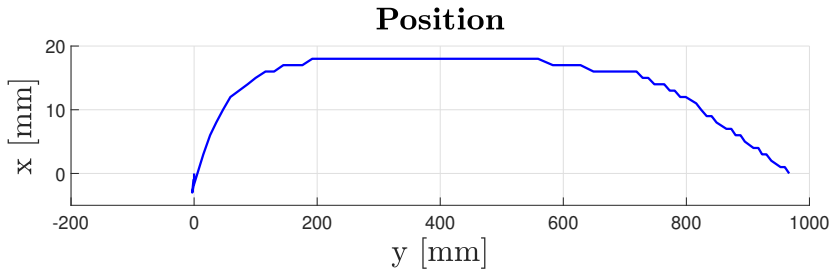
$$\mathbf{K} = \begin{bmatrix} K_{ri,d} \\ K_{ci,d} \end{bmatrix} = \begin{bmatrix} 1.1732 \\ 0.5489 \end{bmatrix} \quad \mathbf{K}_r = \begin{bmatrix} r_{ri} \\ r_{ci} \end{bmatrix} = \begin{bmatrix} 4.899 \\ 5.707 \end{bmatrix}$$

Result



(a) Logged distance to target d_{dt}, reference r_{ri} and output gain u_{ri} at the top figure and linear velocity v_{ri} per sampling interval Δt at the bottom

.



(b) Logged x and y position for the LQR controller without angle alignment

.

fig. 6.8(a) shows the recorded distance to target d_{dt}, reference r_{ri}, output gain u_{ri} and the linear velocity v_{ri} per sampling interval Δt for the LQR controller using only the **Move to target** state. fig. 6.8(b) shows the recorded position for the same test. fig. 6.8(a) shows that d_{dt} steadily decrease to the target coordinate. Secondly, the figure shows how the ramped reference r_{ri} effect the applied output gain u_{ri}. fig. 6.8(a) also show the resulting linear velocity v_{ri}, where it can be observed that there is no change in the value for the 10 first iterations. The same observation can be seen in d_{dt} and indicates that the robot does not move within this time period. This may suggest that the applied output gain does not have any effect for any value below 100. On the contrary, the robot has to rotate 90 degree, indicating the robot is able to rotate at the spot without any considerably deviation in the

position. This is also made clear by the fact that d_{dt} does not increase and have the observed smooth curve. fig. 6.8(b) also shows that the controller is able to apply an output gain that result in a smooth acceleration and de-acceleration of the robot. fig. 6.8(b) shows the logged position of the robot is under 20 millimeters and that the controller produces a smooth curve without any oscillation. It also shows that the position flatten out between the region 200 to 600 $[mm]$ in the y-axis.

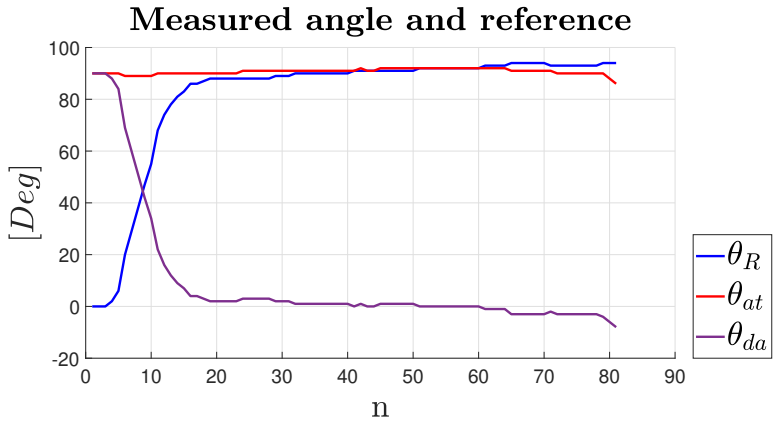


Figure 6.9: Recorded θ_r, θ_{at} and θ_{da} for the LQR controller

.

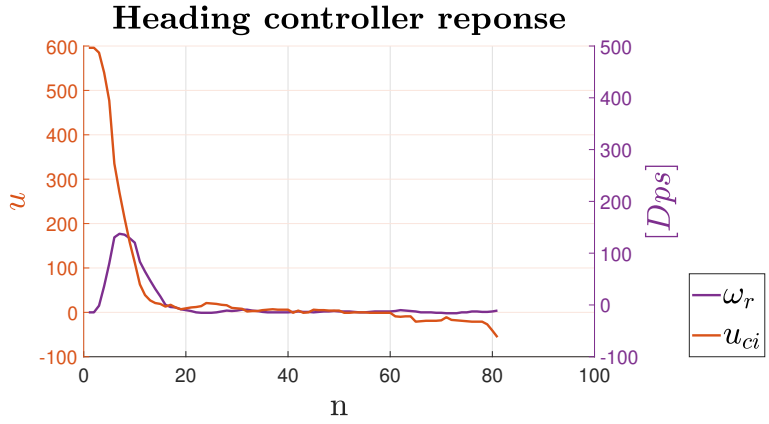


Figure 6.10: Output gain u_{ci} and recorded ω_r for the LQR controller

.

fig. 6.9 and fig. 6.10 shows the response for the heading controller in eq. (6.26) using only the **Move to target** state and θ_{da} directly in the reference $r_{\omega_{ci,k}}$. fig. 6.9 shows the measured heading of the robot θ_R, as well as, the angle to target θ_{at} and the delta angle θ_{da}. Any output gain over $I_T > 250$ will cause one of the wheels to change direction and means that the applied output gain is effectively lower than what is suggested in fig. 6.10. It also allow the robot to turn on the spot.

It can be observed that there is no, to little, osculating in both θ_R and θ_{da} and both reaches a steady state within 20 iteration, corresponding to 1.2 seconds. fig. 6.10 shows a response that is considerable thigh. This is also strengthened by the observation made for d_{dt} and v_{ri} in fig. 6.8(a). All of these observations combined with the resulting curves for θ_R and θ_{da} suggest the heading controller produces an output gain that result in a critical damped response. On the contrary, the small output gain produced from the heading controller between iteration 40 to 60 as shown in fig. 6.9, combined with the observation made for the position in fig. 6.8(b), suggest the presence of steady state error in the position of the robot. This is in some for expected, as the LQR controller does not offer any integral effect that could combat the steady state error.

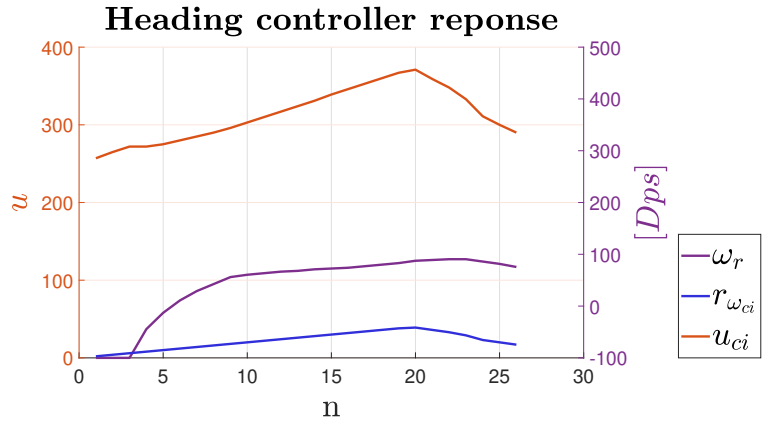


Figure 6.11: Output gain u_{ci}, reference r_{ci} and recorded ω_r for the LQR controller with angle alignment

.

fig. 6.11 shows the response of the angular velocity, the heading reference r_{ci} applying eq. (6.27) and the output gain u_{ci} applying eq. (6.26). The offset seen in u_{ci} is the result of adding a constant of 250, forcing the robot to adjust the heading with opposite direction of the wheels. fig. 6.11 shows the measured heading of the

robot θ_R, as well as, the angle to target θ_{at} and the delta angle θ_{da}. As observed, θ_R is steadily increasing and θ_{da} is steadily decreasing. The robot is stopped within 26 iteration, corresponding to 1.56 seconds and with a final heading of 144 degree.

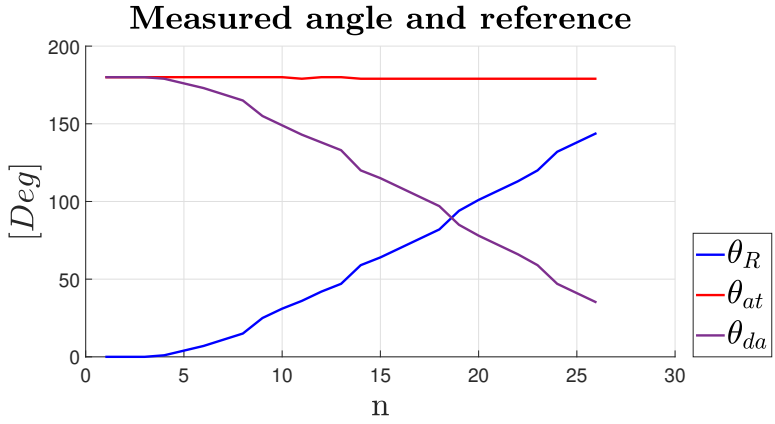


Figure 6.12: Recorded θ_r, θ_{at} and θ_{da} for the LQR controller with angle alignment

.

The observation indicates a fast and smooth response. As θ_{at} is calculated from the difference between the set-point coordinate and the position of the robot using eq. (11), small deviation in the position can cause θ_{at} to switch sign when it is at 180 degree. θ_{at} remains close to constant, an important property for avoiding discontinuous jump in the output gain. Mayor issues with this region with the controller in Arduino IR V1, where the output gain would flip sign and result in the robot rock back and fourth. In contrast, as the reference feed forward state feedback law in eq. (6.26) does not use θ_{at} directly, flipping would only cause the reference r_{ci} to ramp down and decrease the output gain.

6.4 Square test of the new controller

In similar fashion to the square test conducted on Arduino IR V1 described in section 2.4, six square tests are to be conducted on the new implementation. The result from section 6.3 gave insight to the response of the new controller in an isolated manner. The square tests are meant to test the complete system, including the position estimation described in chapter 5. Two 1000×1000 $[mm]$ and two 1500×1500 $[mm]$ square test are carried out for both clockwise (**CV**) and counter clockwise (**CCV**) directions. Two 1000×1000 $[mm]$ alternative square test for **CV** and **CCV** are also conducted for testing how the robot performs without the angle alignment described in section 6.2. This can be considered the most similar

setup compared to the controller found in Arduino IR V1. All of the square tests consist of issuing a series target set-point coordinates to the robot using the values in table 6.2.

Name	SP
Square CCV $1000\ mm$	$[(1000, 0), (1000, 1000), (0, 1000), (0, 0)]$
Square CV $1000\ mm$	$[(0, 1000), (1000, 1000), (1000, 0), (0, 0)]$
Square CCV $1500\ mm$	$[(1500, 0), (1500, 1500), (0, 1500), (0, 0)]$
Square CV $1500\ mm$	$[(0, 1500), (1500, 1500), (1500, 0), (0, 0)]$

Table 6.2: Target set-point coordinates used in the square tests for Arduino IR V2

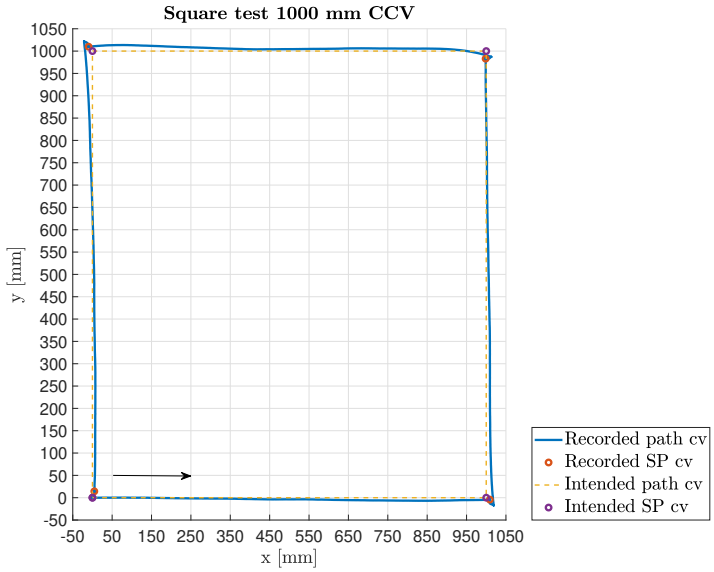
The **Square CCV** and **Square CV** $1000\ mm$ uses a threshold of $15\ [mm]$ and **Square CCV Square CV** $1500\ mm$, as well as, the alternative **Square CCV** and **Square CV** $1000\ mm$ uses a threshold of $30\ [mm]$

Result

fig. 6.13(a) and fig. 6.13(b) shows the result of the $1000 \times 1000\ mm$ square test for counter clock wise and clock wise respectively. fig. 6.14(a) and fig. 6.14(b) shows the result of the $1500 \times 1500\ mm$ square test for counter clock wise and clock wise respectively. Lastly, fig. 6.15(a) and fig. 6.15(b) shows the result of the $1000 \times 1000\ mm$ for the alternative controller in counter clock wise and clock wise respectively.

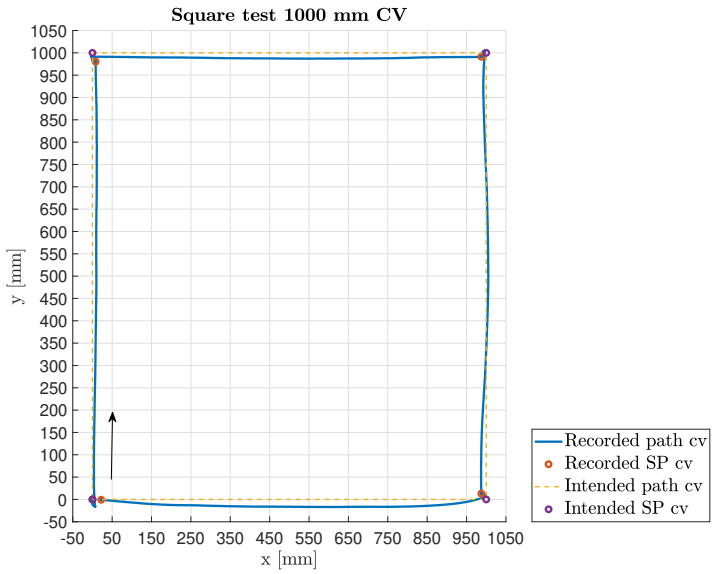
CCV $1000\ mm$	x	1009.12	998.45	-9.76	4.69
	y	-4.28	983.01	1009.84	14.27
CV $1000\ mm$	x	8.12	987.69	987.43	22.25
	y	980.00	991.15	13.15	-0.73
CCV $1500\ mm$	x	1456.64	1457.48	-38.15	-33.28
	y	-2.26	1457.124	1487.92	-14.22
CV $1500\ mm$	x	-12.55	1438.97	1442.29	-14.14
	y	1440.30	1455.33	-38.95	-57.33
CCV $1000\ mm$ alt	x	997.21	990.90	-3.28	-1.29
	y	0.02	969.05	967.57	-4.14
CV $1000\ mm$ alt	x	5.00	962.04	969.32	19.04
	y	968.31	987.01	3.58	-8.12

Table 6.3: End point result for each square test



(a) Square test counter clock wise

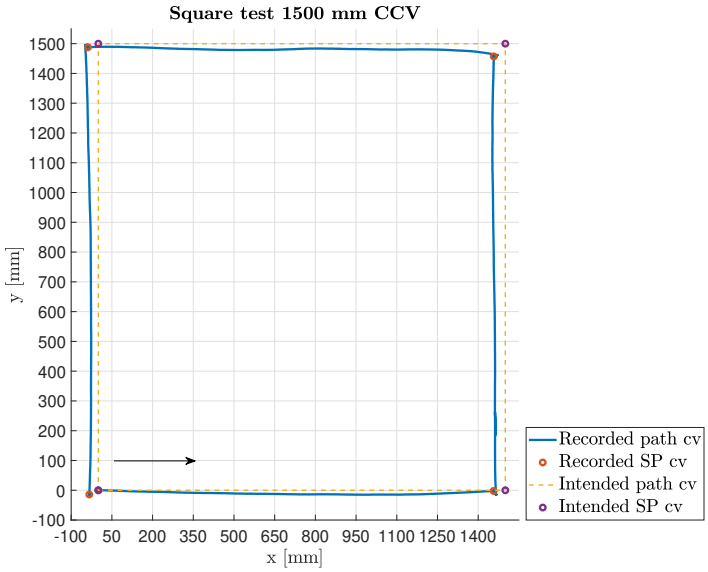
.



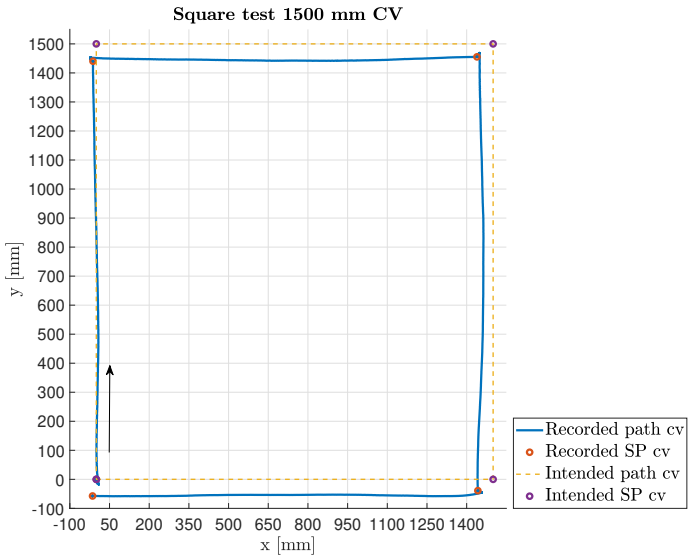
(b) Square test clock wise

.

Figure 6.13: Square test 1000 mm with the use of the implemented LQR controller

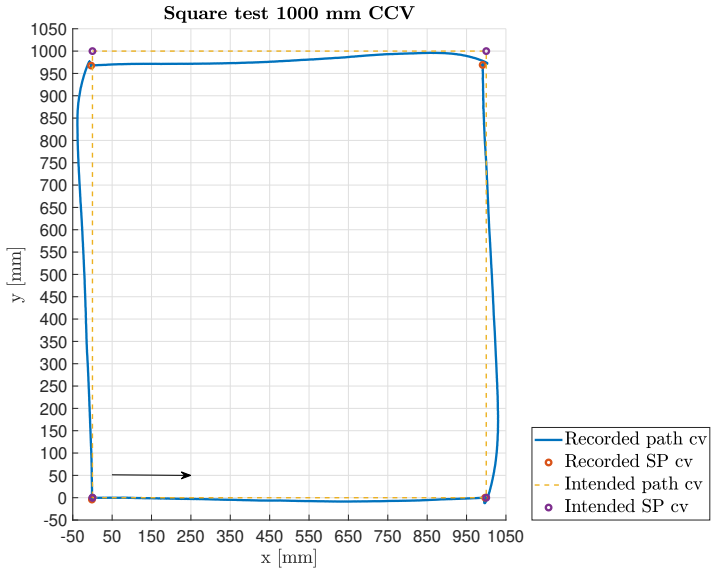


(a) Square test counter clock wise



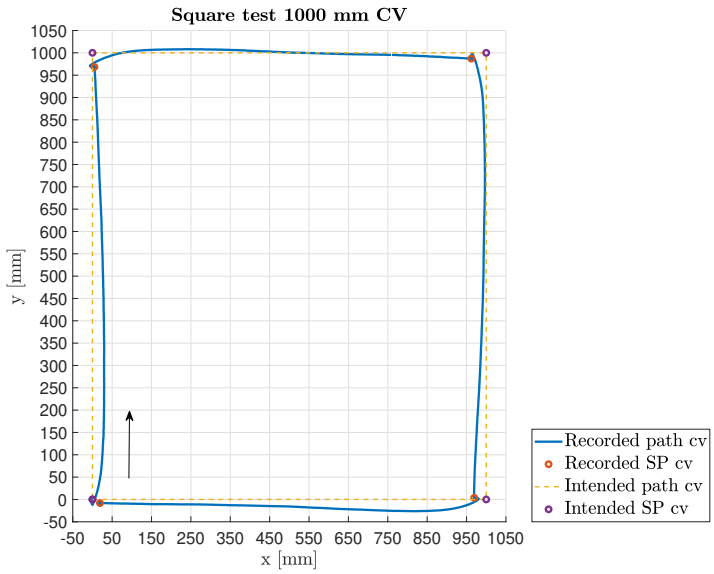
(b) Square test clock wise

Figure 6.14: Square test 1500 mm with the use of the implemented LQR controller



(a) Square test counter clock wise

.



(b) Square test clock wise

.

Figure 6.15: Square test 1000 mm with the use of the implemented LQR controller and without angle alignment

As seen from fig. 6.13(a) and fig. 6.13(b) the LQR controller produces a path that resemble a perfect line. For a perfect line, the robot would have followed a path similar to the yellow dotted line. Observation made from the result in fig. 6.13(a) and fig. 6.13(b) reviles that the overall deviation between the line and the path of the robot is in the range of $10 - 20$ millimeters. The deviation observed in both of the 1500×1500 shown in fig. 6.14(a) and fig. 6.14(b) have a similar range. The overall deviation for the LQR controller without angle alignment, shown in fig. 6.15(a) and fig. 6.15(b) is within the range of $20 - 30$ millimeters. The overall deviation is consistent with the deviation logged on the robot as shown in fig. 6.8(b). The LQR controller does does not align the robot perfectly with the target coordinate when using the angle alignment in **Align to target**. Nevertheless, the observed path for each test in combination with the response from fig. 6.9, shows that the controller offer a response that tightly correct the heading of the robot. In addition, the controller provide a similar response when controlling the motion of the robot exclusively in the **Move to target** state as shown in fig. 6.15(a) and fig. 6.15(b). The combined result shows that the LQR controller offer a very tight and smooth control for guiding the robot to a target coordinate.

A consistent trend observed from the end-point result in table 6.3 is that the end-point values are below the target threshold when the robot is rotated ± 90 degrees. An aspect to consider given this observation, is that the position of the robot is tracked using markers. The recorded position calculated at a center point from all of the markers. Given the shape of the robot as seen in fig. 3.1(a), there is an indication that there is an offset between the actual center of rotation for the robot and the recorded one. The presence of the offset is strengthened by by the fact that the recorded position is consistently shifted 20 millimeters back when the robot is rotating ± 90 degrees. The result from fig. 6.12 shows that θ_{at}, found from eq. (2.16), does not change considerable when the robot is alignment to the target, meaning the the position of the robot does neither change considerably when rotating. Assuming the presence of the offset, the end-point result in table 6.3 would be within the given target threshold. However, there is still error in the position estimate. The result from section 5.4 gave an indication that the mean error in the position estimate is within the range of $10 - 15$ millimeters. It also showed that the accelerometer made the error in the position estimate stay within a region of the actual position. This is in contrast with the constant offset error occurring when only using the encoders. The end point result from table 6.3, in addition to, the result in fig. 6.13, fig. 6.14 and fig. 6.15 is consistent with the observation made in section 5.4 about the error and the use of the accelerometer.

Over several runs, the wheel will accumulate dirt. However, the accumulation

seems to saturate, resulting in a initial decrease in the traction of the wheels. This changes the condition in eq. (6.14) and the maximal acceleration of the robot. The ramping of the reference r_{ri}, as well as, target alignment limits the maximum acceleration the robot can experience. Still, the target alignment can pose an issue with the position estimate as the robot rotates on the spot. This means that the accelerometer cannot combat the error from the encoders. The reduce traction can result in the robot overshooting the target angle followed by an output gain that result in a sudden change in the direction of the robot. The sudden change can lead to wheel slip and resulting error in the position estimate.

Regardless, using reference feed forward with state feedback for controlling the motion of the robot is a solid improvements comparing to the control system found in Arduino IR V1. Achieving the same result with the "black box" PID controller found in Arduino IR V1 would have been far more difficult. Applying the same output gain in the old system would have resulted in an opposite large output gain and decreasingly smaller ripples. As the LQR controller uses state feedback, it seems to indirectly predict the movement of the robot by utilize the angular velocity. Using LQR for finding the state gain $\mathbf{K}$ also provide an automatic tuning of the system, as most of the work was in the development of the dynamic model in eq. (6.23).

Chapter 7

Path planning and object avoidance

The current system is entirely dependent on the main server for safely navigating the robot within an environment as described in section 2.1. Relying entirely on the server for making good decision is, in the perspective of fault tolerance, an inadequate approach as the robot is not able to operate without the main server. This is combined with several issues observed from the server side. The issues are first and foremost in the form of un-handled exceptions that result in the server being unresponsive. Another problem is that the real position of the robot cannot be exactly known and drifting cannot be entirely eliminated, even thought the result from chapter 5 have shown to give good accuracy in the position estimate. This poses a problem, especially in asymmetric environments, where divergence between the position known to the server and the real position of the robot can cause the robot to crash into an object by an ill-advised target set-point coordinate.

It is still possible to develop a path planning procedure feasible to implement on the robot that makes it less dependent on a stable operation of the server. The use of artificial potential field is a potential method, that is based on constructing a potential field $U(q)$ with an attractive component $U_{att}(q)$ and a repulsive component $U_{rep}(q)$ on a given configuration space q based on eq. (7.1) [9].

$$U(q) = U_{att}(q) + U_{rep}(q) \tag{7.1}$$

The robot in the configuration space is treated as an point particle and the attractive field is used to guide the robot to a point q_{final}. The repulsive field U_{rep} is used to repel the robot from obstacles.[9]. However, the highly limited computational resources found on the robot is a mayor factor that restrict how the artificial potential field method is implemented. The robot cannot store or base a search on the whole

configuration space with the complete field U. Therefore a method designed from scratch needs to be implemented.

The proposed solution is to consecutive construct a field from the scanning the environment using the sensor tower and the current measurement from the IR sensors. Information about the environment is extracted by combining the IR measurement with the current position and heading of the robot. Based on this information, an algorithm is developed to determine if the robot can drive directly to the initial target set-point coordinate or constructing a series of augmented coordinates for avoiding colliding with objects in the environment. The motion controller described in chapter 6 provide the attractive field as the result from section 6.4 has showed that the robot traverse a given path in a fairly straight line with only small deviations. The procedure is to be run in combination with the algorithm found in the main server. This way the server can focus on finding an optimal target set-point coordinate at the edge of the discovered map and the robot can find a path to that coordinate that is safe to traverse. Given a initial target set-point coordinate from the main server, the robot utilize the four IR sensors to find the location of an object relative to the robots current position and heading. Based on this, an intermediate or augmented set-point coordinate is found such that the robot does not collide with the object. A series of these augmented set-point coordinates can be then connected to form a path to the original target.

The chapter is structured in the following way:

- Processing and filtering of the IR sensors section 7.1
- Description of Path planning procedure section 7.2
- Task implementation of the Path planning procedure section 7.3
- Evaluation of the Path planning procedure section 7.4

7.1 Sensors processing and filtering

As the path planning procedure realises on the distance measured from the the IR sensors, an accurate mapping from the measured voltage to a physical distance is needed. The intensity from the IR measurement is not linear with the distance. and the raw data from a IR sensor is inverted. A mapping proposed by *Navarro, Benet & Blanes* (Conference paper 2008) that is based on the inverse square law is recited in eq. (7.2) *where V is the measured voltage, α is a parameter for the IR intensity and η is the angle incident to the measured target* [6].

$$d = \sqrt{\frac{\alpha}{V}}\sqrt{cos(\mu)} \tag{7.2}$$

However, the model in eq. (7.2) does not exactly fit the measurement from the Sharp G2D12 IR sensors. For that reason, an own model was developed based on a modified version of eq. (7.2), as well as, empirically fitment of observed IR measurements. The model was found to be

$$d_{IR}(\propto) = K_{IR} L_{unit} \left[\sqrt{\frac{1}{\propto}} - 0.1 \right], \ \propto > 0 \tag{7.3}$$

where $\propto$ is a scaled intensity in the range of $[\propto_{min}, 100]$, L_{unit} is a gain for scaling the measurement in the range of $[0, L_{unit}]$ and K_{IR} is a correction gain. The constant of 0.1 is added such that

$$d_{IR}(100) = 0$$

The model was implemented into the system with a max and min saturation using eq. (7.4).

$$d_{IR}(\propto) = \begin{cases} 0 & \propto > L_{unit} \\ L_{unit} & \propto < \propto_{min} \\ K_{IR} L_{unit} \left[\sqrt{\frac{1}{\propto}} - 0.1 \right] & \propto_{min} \leq \propto \leq L_{unit} \end{cases} \tag{7.4}$$

7.2 Method definition

Objects in front of the robot is most important to detect for determining, with respect to the current position and heading of the robot, if the current path is safe to traverse or some form of re-routing is required. The measurement from the forward and right IR sensor are used to construct a scan field $\mathbf{F}$ defined by

$$\mathbf{F} = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_{j-1} \\ f_j \end{bmatrix} \tag{7.5}$$

where the size of the vector is the scan resolution R_s. The scan field forms a horizontal semi circle of 180 degree around the forward direction of the robot. This is carried out by successive sampling the right and forward IR sensor and rotating the servo with an angle θ_s. The right sensor start at index 0 and the forward sensor starts at an index $R_s/2$ that is proportional to a 90 phase shift, as shown in fig. 7.1. Combining the measurement from the right and forward sensors means that the sensor tower only need to rotate 90 degree to complete a 180 degree scan.

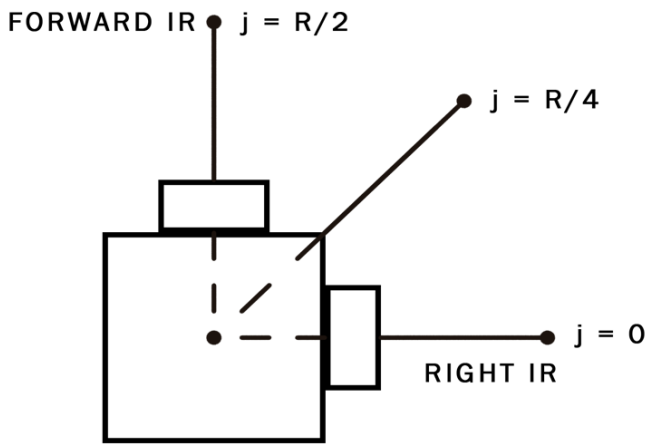


Figure 7.1: Index mapping for the scan field using the forward and right IR sensor

.

An angle θ_f is mapped to each sampling point using the LinMap function described in section 2.3. The defined range of θ_f is $[-90, 90]$, where an index of 0 correspond to an angle of -90, as well as the initial location of the right IR sensor. The initial location of the forward sensor is at 0 degree. Given a complete scan, the minimum of $\mathbf{F}$ can be used with eq. (7.3) in combination with θ_f to find a coordinate within the world frame using eq. (7.6).

$$\begin{bmatrix} x_{k+1}^w \\ y_{k+1}^w \end{bmatrix} = \begin{bmatrix} x_k^w + d_{IR}\cos(\theta_f) \\ y_k^w + d_{IR}\sin(\theta_f) \end{bmatrix} \tag{7.6}$$

However, using solely $\mathbf{F}$ is not adequate to construct a augmented coordinate that is safe to traverse. Therefore a weighted sum defined by eq. (7.7)

$$z_k = \mathbf{w} \cdot \mathbf{x} \tag{7.7}$$

is used to construct a vector $\mathbf{z}$ defined by eq. (7.8) with size $R_s \times 1$

$$\mathbf{z} = \begin{bmatrix} z_0 \\ z_1 \\ \dots \\ z_{j-1} \\ z_j \end{bmatrix} \tag{7.8}$$

and the element of $\mathbf{z}$ with the minimum sum using eq. (7.9) is used to find a sampling point of F to construct the augmented set-point target coordinate.

$$i = \min(\mathbf{z}) \tag{7.9}$$

The weighted sum in eq. (7.7) uses the weight vector $\mathbf{w}$ with elements $[w_\Gamma, w_\Theta, w_P]$ and the three state vectors $\mathbf{\Gamma}$ $R_s \times 1$, $\mathbf{\Theta}$ $R_S \times 1$ and $\mathbf{P}$ $R_S \times 1$ defined in eq. (7.10) is used as features of the obtained scan field $\mathbf{F}$.

$$\mathbf{\Gamma} = \begin{bmatrix} \gamma_0 \\ \gamma_1 \\ \vdots \\ \gamma_{j-1} \\ \gamma_j \end{bmatrix} \quad \mathbf{\Theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_{j-1} \\ \theta_j \end{bmatrix} \quad \mathbf{P} = \begin{bmatrix} \rho_0 \\ \rho_1 \\ \vdots \\ \rho_{j-1} \\ \rho_j \end{bmatrix} \tag{7.10}$$

The features are used to take into account constraint poses by the robot and the environment. They relies on information obtained by the scan field $\mathbf{F}$ in combination with the current position and heading of the robot, as well as the distance and angle to a given target coordinate. $\mathbf{\Gamma}$ contains a normalized IR measurement, $\mathbf{\Theta}$ contains linear increasing values corresponding to the angle to target coordinate. $\mathbf{P}$ contains a proxy value indicating if a scaled IR measurement using eq. (7.3) have a radius less than the target radius.

The angle to target feature $\mathbf{\Theta}$

The $\mathbf{\Theta}$ feature is constructed by mapping a given index j in the range $[0, j_{end}]$ to a weight in the range of $[-100, 100]$

$$\theta_k = \text{LinMap}(j, [0, j_{end}], [-100, 100]) - \phi \tag{7.11}$$

where ϕ is found by a pivot index j_p

$$\begin{aligned} \phi &= \text{LinMap}(j_p, [0, j_{end}], [-100, 100]) \\ j_p &= \text{LinMap}((\theta_{at} - \theta_w), [-90, 90], [0, j_{end}]) \end{aligned}$$

The proxy to target feature $\mathbf{P}$

$\mathbf{P}$ contains a value for each sampling point indicating if a sampling point is larger than a radius equal to the distance to target

$$w_{ad,k} = \frac{1}{(\frac{d_{dt}}{K_{IR} L_{unit} + 0.1})^2} \tag{7.12}$$

and

$$\rho_k = \begin{cases} 0 & (f_k - w_{ad,k}) < 0 \\ \frac{100}{d_{dt} L_{unit}} & (f_k - w_{ad,k}) \geq 0 \end{cases} \tag{7.13}$$

The normalized scan field $\mathbf{\Gamma}$

The normalization of $\mathbf{F}$ to obtain $\mathbf{\Gamma}$ is carried out by using discrete convolution for a finite set defined by eq. (7.14) [4].

$$(\Gamma * k)[n] = \sum_{m=0}^{K} k[m] \cdot \Gamma[n-m] \tag{7.14}$$

The convolution is normally used between two signals, but it can also be used with a signal and a kernel k with size $1 \times K$ and the weights

$$\mathbf{k} = [k_0, k_1, \ldots, k_{i-1}, k_i] \tag{7.15}$$

as well as normalizing the discrete convolution with k, resulting in the definition

$$(\Gamma * k)[n] = \sum_{m=0}^{K} \frac{1}{K} k[m] \cdot \Gamma[n-m] \tag{7.16}$$

For obtaining $\mathbf{\Gamma}$, a kernel size of three is used with the elements $\mathbf{k} = [k_0, k_1, k_2]$. The convolution is implemented on the Atmega2560 MCU represented by the pseudo code in algorithm 7 that handles the edges of the scan field using a slide index.

Algorithm 7: Normalized convolution

```
Make temporary array T of F
n ← length of F
m ← length of kernel k
for i = 0 TO (n-1) do
    for i = 0 TO (m-1) do
        slideindex = n − m + 1
        if slideindex < 0 then
            slideindex = 0
        else if slide index = (length of F − 1) then
            slide index = (length of F − 1)
        end if
        T[n + 1] = T[n] + k[m] · F[slide index]
    end for
    T[n] = T[n]/K
end for
Copy content from T to Γ
```

Finding the minimum of z

The system only need to store the scan field $\mathbf{F}$ which is transformed to $\mathbf{\Gamma}$ using eq. (7.16). The vector $\mathbf{\Theta}$ is found by eq. (7.11) and $\mathbf{P}$ is found using eq. (7.13). For the path planning procedure the explicit weighted sum in eq. (7.17) is used.

$$z_k = \frac{100(w_\Gamma \gamma_k + w_\Theta \theta_k - w_P|\rho_k| - 0.2\gamma_{thrs,k})}{3} \tag{7.17}$$

Algorithm 8: Finding the minimum of z_k

```
Obtain Γ using algorithm 7
for j = 0 TO R do
    Compute θ_j using eq. (7.11)
    Compute ρ_j using eq. (7.13)
    Compute z_j using eq. (7.17)
    if z_o < z_1 then
        z_1 = z_0
        store index j as the current j_min
    end if
end for
```

The vector z_k is also memory optimized by using a vector for two elements $\mathbf{z} = [z_0, z_1]$ where z_1 is replaced by z_0 for finding the minimum. algorithm 8 shows a pseudo code of the method for finding the minimum sampling point.

Constructing the augmented target set-point coordinate

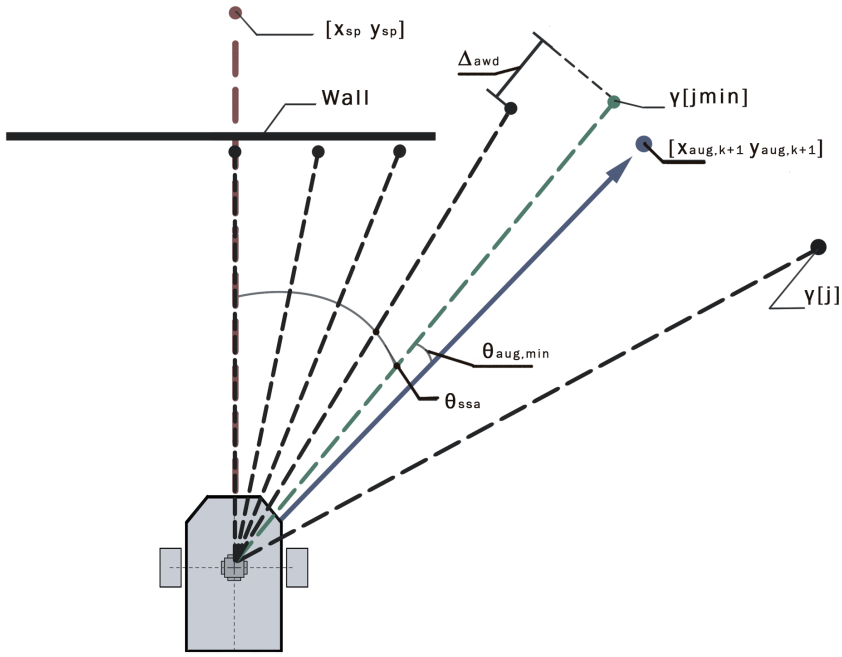


Figure 7.2: Illustration of variables used for calculating the augmented target set-point coordinate based on the scan field

The index containing the minimum of $\mathbf{z}$ is used to construct the augmented target set-point coordinate, which is found using eq. (7.6) and a augmented minimum angle and distance. The mapping of the scan step angle θ_{ssa} for each index in the vector $\mathbf{z}$ in combination with Δ_{awd}, corresponding the change in the scan field, is utilized to find the augmented minimum angle. The augmented distance applies eq. (7.3) based on the sample point corresponding to the inverse mapping of the augmented minimum angle to a scan field index. The scan step angle θ_{ssa} is found by hashing the the minimum index j_{min} using

$$\theta_{ssa} = \begin{cases} 0 & j_{min} = R/2 \\ \text{LinMap}(j_{min}, [0, j_{end}], [-90, 90]) & j_{min} \neq R/2 \end{cases} \tag{7.18}$$

The angle weight direction Δ_{awd} found by eq. (7.19) is used to push θ_{ssa} in either

a left or right direction based on the shape of the scan field. Δ_{awd} is computed by the difference between the min sample point and a left and right sample point given by R_{dsi}.

$$\Delta_{awd} = \begin{cases} \gamma[j_{min} - R_{dsi}] - \gamma[j_{min}] & j_{min} = R - R_{dsi} \\ \gamma[j_{min}] - \gamma[j_{min} + R_{dsi}] & j_{min} = R_{dsi} \\ \gamma[j_{min} - R_{dsi}] - \gamma[j_{min} + R_{dsi}] & j_{min} \end{cases} \tag{7.19}$$

The augmented angle $\theta_{aug,min}$ is found by eq. (7.20), limited to the range $[-\pi, \pi]$ and where $K_{dist,rep}$ is the repulsive angle gain.

$$\theta_{aug,min} = \theta_{ssa} + K_{dist,rep} \arctan\left(\frac{\Delta_{awd}}{8}\right) \tag{7.20}$$

As $\theta_{aug,min}$ no longer correspond to the minimum sample point, a heading reflection index j_{hfi} is computed by eq. (7.21).

$$j_{hfi} = \text{LinMap}(\theta_{aug,min}, [-90, 90], [0, j_{end}]) \tag{7.21}$$

and the corresponding sample point from $\mathbf{\Gamma}(j_{hfi})$ is used with eq. (7.3) and the values $K_{IR} = 1.65$ and $L_{unit} = 400$ to find the augmented distance

$$d_{aug} = d_{IR}(\mathbf{\Gamma}(j_{hfi})) - K_{dist,rep} \tag{7.22}$$

where $K_{dist,rep}$ is the repulsive distance gain. The augmented distance d_{aug} together with $\theta_{aug,min}$ is used to construct the augmented target set-point coordinate

$$\begin{bmatrix} x^w_{aug,k+1} \\ y^w_{aug,k+1} \end{bmatrix} = \begin{bmatrix} x^w_k + d_{aug}\cos(\theta_{aug,min}) \\ y^w_k + d_{aug}\sin(\theta_{aug,min}) \end{bmatrix} \tag{7.23}$$

Overall the method is fairly simple and offer a low computation cost considering this is to be run on a embedded microcontroller with highly limited resources. algorithm 9 shows the complete algorithm used for finding the augmented set-point coordinate.

7.3 Path planner task

The Path planner task is designed to offer two scan modes; augmented scan used for algorithm 9 and a ordinary scan using all of the four IR sensors. The Path planner task uses a state based design with the four states;

- **Idle**: Wait and poll scan request in the form of FreeRTOS events.

Algorithm 9: Finding an augmented target coordinate

Scan the environment for obtaining $\mathbf{F}$
Use algorithm 8 to find the minimum index j_{min}
Find θ_{ssa} using eq. (7.18) with j_{min}
Find Δ_{awd} using eq. (7.19) with j_{min}
Compute $\theta_{aug,min}$ using eq. (7.20) with θ_{ssa} and Δ_{awd}
Find d_{aug} using $\mathbf{\Gamma}(j_{hfi})$ in eq. (7.3)
Based on $\theta_{aug,min}$ and d_{aug}, find augmented target set-point coordinate using eq. (7.6)

- **Ordinary scan**: Sample the IR sensor and map the measurement to a distance using eq. (7.3), as well as, put mapped distance into a buffered FreeRTOS queue and send update to the Server communication task for each scan step.
- **Augmented scan**: Construct scan field $\mathbf{F}$ using the right and forward IR sensor.
- **Compute augmented target**: Compute an augmented target coordinate using algorithm 9 based on the obtained $\mathbf{F}$.

The computation of an augmented target set-point coordinate is performed when a new target set-point coordinate is received from the server. The task will continue to compute augmented target set-point coordinate until the original target is reached. An ordinary scan is performed when the robot is connected to the server and each time the robot reaches a target set-point coordinate. For computing an augmented target set-point coordinate, the Path planner task require a position update produced from the Estimator task and a series of motion operation provided by the LQR controller task on Node 1.

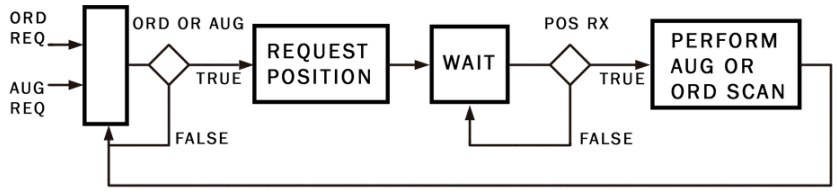


Figure 7.3: Block diagram of the two scan mode on the Path planner task

.

fig. 7.3 shows the operation for performing an ordinary or augmented scan. The task returns to the **Idle** state after an ordinary scan. After an augmented is com-

pleted, the current position of the robot is compared with the original target set-point coordinate. If the position is under a given threshold, the task goes to the **Compute augmented target** state, where a new augmented target set-point coordinate is calculated. The Path planner task work along side the Coordinator task, which decide when to perform an ordinary or augmented scan. The coordinator task issue the following request for computing an augmented target set-point coordinate:

1. Send received target from server and Start signal to the LQR controller.
2. When the robot is aligned to target: Send reset signal to LQR controller and augmented scan request to the Path planner, which send in return an internal position request.
3. Forward position request to Position estimator task.
4. When new position update received: Set position updated event.
5. When augmented target computed: Forward computed augmented target and Start signal to the LQR controller.
6. When robot is aligned to augmented target: Send Start signal to the LQR controller.
7. When robot have reached the augmented target: Send a ordinary scan request to the Path planner task.
8. When ordinary scan complete: Repeat step 2 to 7 until the robot have reached the server target coordinate.

7.4 Evaluation of the Path planner task

eq. (7.4) is used for testing the mapping of the four IR sensors. The values in table 7.1 for K_{IR} and L_{unit} is used in the mapping.

K_{IR}	5.4
L_{unit}	800
$\propto_{min}$	14

Table 7.1: Parameters used for the IR mapping

Three test are used for the Path planner procedure; the first is to confirm the construction of the scan field and implemented algorithm in algorithm 9 for finding an augmented target coordinate. The second test is concerned with confirming if the robot is able to safely navigate the environment for an issued target set-point coordinate. And lastly, the mapping and navigation capabilities of the robot is tested. The parameters in table 7.2 is used for testing the Path planner procedure

Kernel	$[1.05, 1.20, 1.05]$
R_s	40
$K_{\theta,rep}$	0.44
$K_{dist,rep}$	100
R_{dsi}	5
$\mathbf{w_z}$	$[1.2, 0.06, 0.15, 0.2]$

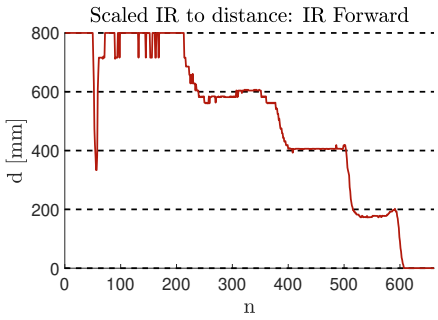
Table 7.2: Parameters for obtaining the augmented target set-point coordinate

Testing the IR mapping

To test the model, markers equal to $[800, 600, 400, 200, 0]$ mm where measured and a plate was placed on the respective markers and the calculated distance was logged.

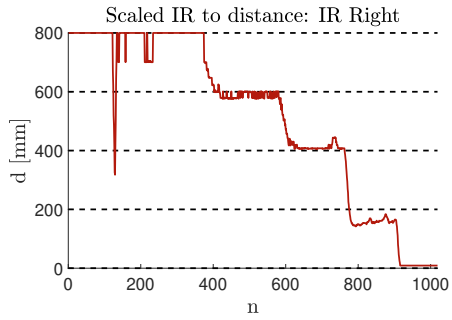
Result

fig. 7.4(a),fig. 7.4(b),fig. 7.5(a) and fig. 7.5(b) shows the resulting mapping of the scaled intensity $\propto$ to the distance $[800, 600, 400, 200, 0]$ mm. As shown, it is able to give a reasonable mapping for all of the sensors using the same K_{IR} and L_{unit} for all sensors.



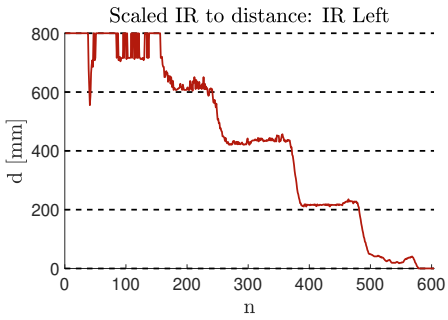
(a) Mapped raw IR measurement to distance in mm for the forward IR sensor

.



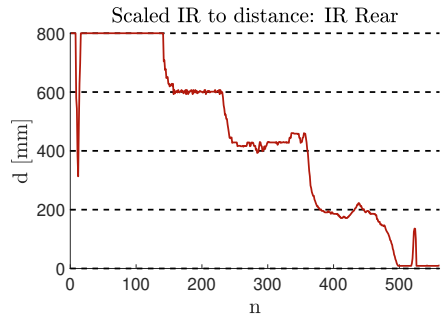
(b) Mapped raw IR measurement to distance in mm for the right IR sensor

.



(a) Mapped raw IR measurement to distance in mm for the left IR sensor

.

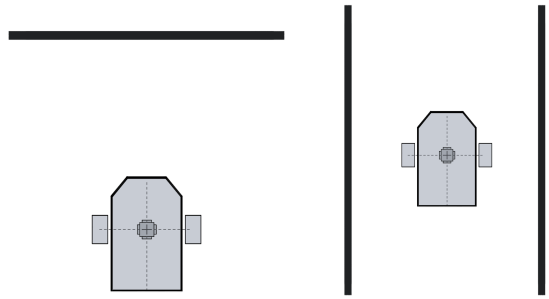


(b) Mapped raw IR measurement to distance in mm for the rear IR sensor

.

Path planner scans

fig. 7.6(a) and fig. 7.6(b) gives a 2D illustration of the two common scan scenarios. The first is the robot facing a wall and the second is two walls located on the side of the robot. For both scenarios, the position and the heading of the robot in the world frame is $(0, 0)$ mm and 0 Deg respectively and a target set-point coordinate of $(500, 0)$ is issued to the robot.

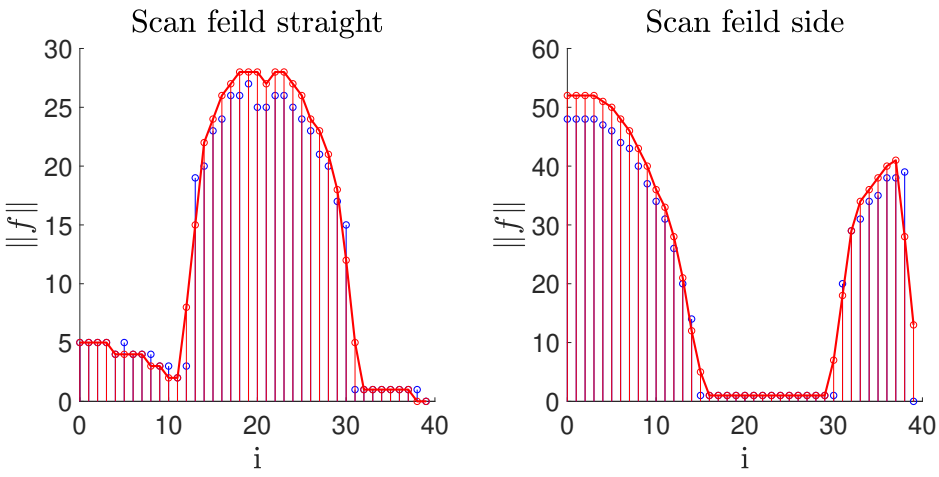


(a) Straight **(b)** Side

Figure 7.6: Illustration of two different scan scenarios

Result

fig. 7.7(a) and fig. 7.7(b) shows the obtained scan field and the normalized field for the two scenarios. By inspections of fig. 7.7(a) and fig. 7.7(b) it is clear that the intensity for the scaled IR measurement can provide the properties for deciding a sample point for finding the augmented target set-point coordinate. For both the straight and side wall scenario, the intensity is high for scans close to a wall and low for open or free spaces. fig. 7.7(a) and fig. 7.7(b) also shows the smoothing produced by the convolution.



(a) Obtained scan field for the straight wall scan. **(b)** Obtained scan field for the side wall scan

Figure 7.7: Blue is the raw scan field and the normalized field in red.

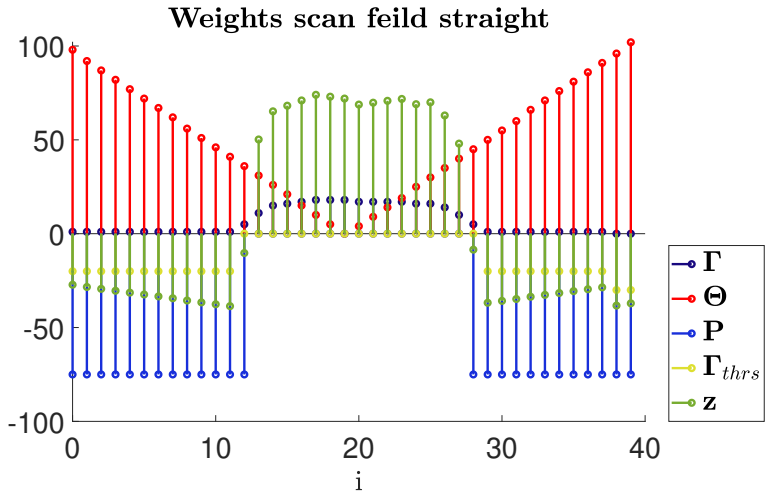


Figure 7.8: Obtained features and weighted sum for the straight wall scan

fig. 7.8 shows the elements from the four features $\mathbf{\Gamma}$, $\mathbf{\Theta}$, $\mathbf{P}$ and $\mathbf{\Gamma}_{thrs}$, in addition to the elements from each weighted sum in $\mathbf{z}$ for the straight wall scan. As mention previously, $\mathbf{\Gamma}$ is the normalized scan field, containing the scale IR intensity measurement. The proxy to target $\mathbf{P}$ contains the mapped distances of the elements in $\mathbf{\Gamma}$ using eq. (7.3). As the $\mathbf{P}$ favour any elements that have a radius less than the

distance to target relative to the center point of the robot, points on both the right and left side of the robot have a strong negative value. The feature $\mathbf{\Gamma}_{thrs}$ is also helps favouring open space. In this scenario it is not that relevant, as the target set point coordinate have a relative short distance. As shown in fig. 7.8 there exists sample points that have distances larger than 500 millimeters on the left and right side. $\mathbf{\Gamma}_{thrs}$ is mainly used for cases where the distance to target is larger than the maximal range of 800 millimeters for the IR sensors. As the relative position of the robot is zero in the world frame, the resulting angle to target θ_{at} is zero for the target coordinate $(500, 0)$. As shown from fig. 7.8, $\mathbf{\Theta}$ contains linear increasing elements that penalize sample points that are further away from θ_{at}. As shown in the illustration in fig. 7.1, the resulting pivot index j_p is 20, corresponding to the forward direction of the robot. Therefore, the feature $\mathbf{\Theta}$ attracts the robot to the target coordinate.

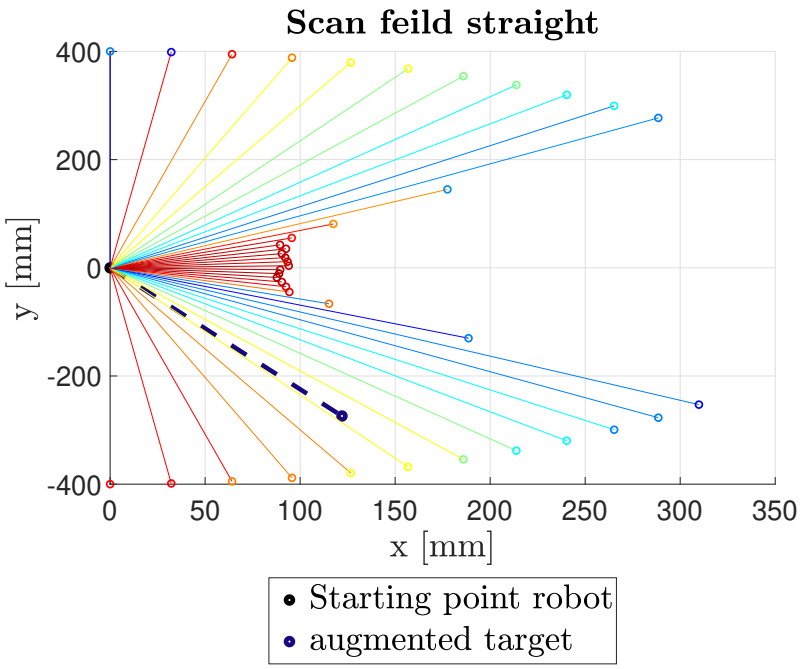


Figure 7.9: 2D visualization of the straight wall scan

.

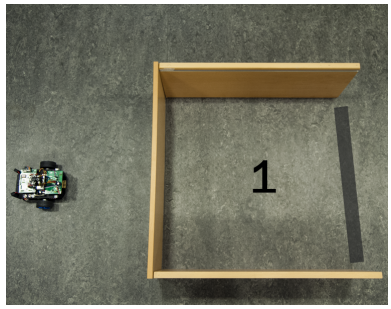
fig. 7.9 shows a 2D visualization of each scanned sample points mapped to a distance using eq. (7.3) and the parameters in table 7.1 for the straight wall scenario.

The colors visualize each element of $\mathbf{z}$. As seen from fig. 7.8, both the left and right side of the wall have the lowest sum of $\mathbf{z}$. The resulting min index j_{min} found from the scan, running the algorithm algorithm 8 on the robot was index 11. The step angle θ_{ssa} corresponding to the angle of the min sample point found from eq. (7.18). For the index $j_{min} = 11$ result in $\theta_{ssa} = -39$ degree. The initial angle of θ_{ssa} is pushed further to the left using Δ_{awd}, based on the slope of the scan field in fig. 7.7(a). The magnitude of Δ_{awd} is damped by the slope in the right side of the min index, resulting in an augmented minimum angle $\theta_{aug,min}$ of -66 degree. Together with a augmented min distance of 300 millimeters and eq. (7.6), the resulting augmented set point coordinate is

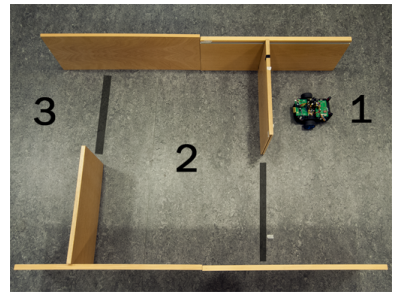
$$\begin{bmatrix} x^w_{aug,k+1} \\ y^w_{aug,k+1} \end{bmatrix} = \begin{bmatrix} 122\ [mm] \\ -274\ [mm] \end{bmatrix}$$

As shown in fig. 7.9 and the corresponding augmented target set-point coordinate $(122, -274)$ confirms the described algorithm listed in algorithm 9.

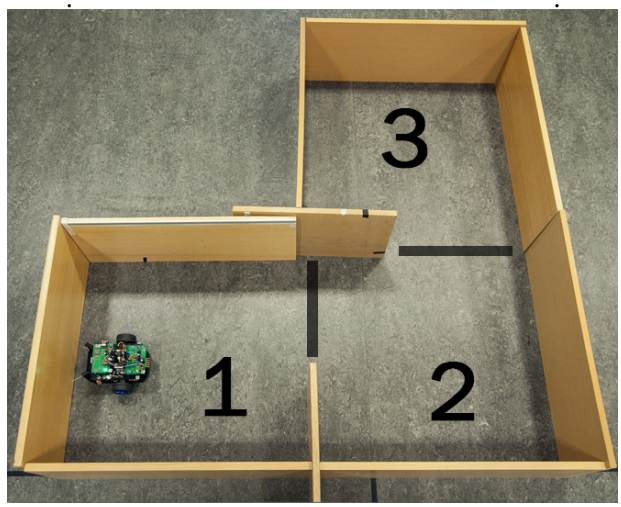
Path planner courses



(a) Box course



(b) Slalom course



(c) Final course

Figure 7.10: Different courses fro the testing the path planner procedure

fig. 7.10 illustrate the three different courses used for testing the path planner procedure. The goal of the robot is to find a path in the environment such that it reaches a given target coordinate without crashing into the walls. fig. 7.10(a) shows the Box course, where the robot starts on the outside and a target set-point coordinate of $(1000\ ,\ 0)$ is issued to the robot. The coordinate is within the box, marked **1**. The black line represent the opening or gate into the designated area. fig. 7.10(b) shows the Slalom course, where a target set-point coordinate of $(2000\ ,\ 0)$ is issued. The robot has to traverse three sectors marked **1**, **2** and **3**. fig. 7.10(c) shows the Final course, where two target set-point coordinates, $(1000\ ,\ 0)$ and $(1000\ ,\ 1000)$, is issued to the robot. Likewise, the robot has to traverse three

restricted sectors marked **1**, **2** and **3**, in addition to reroute 90 degree.

Result

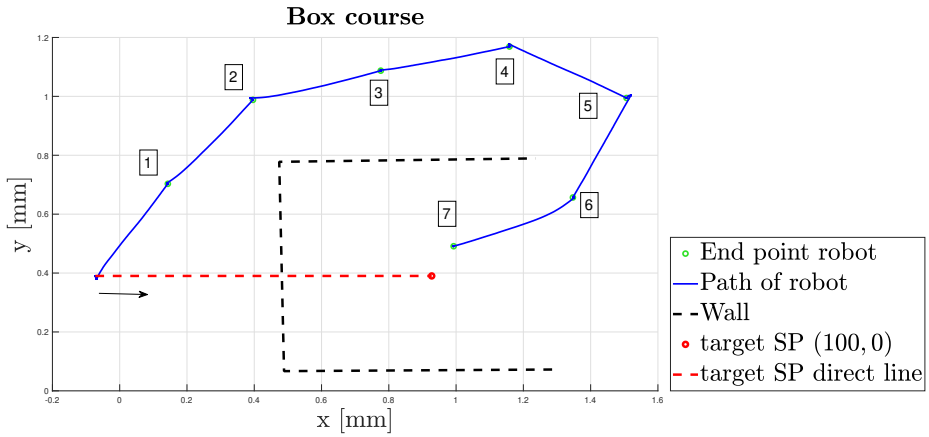


Figure 7.11: Running the path planner procedure for finding the path in the Box course

.

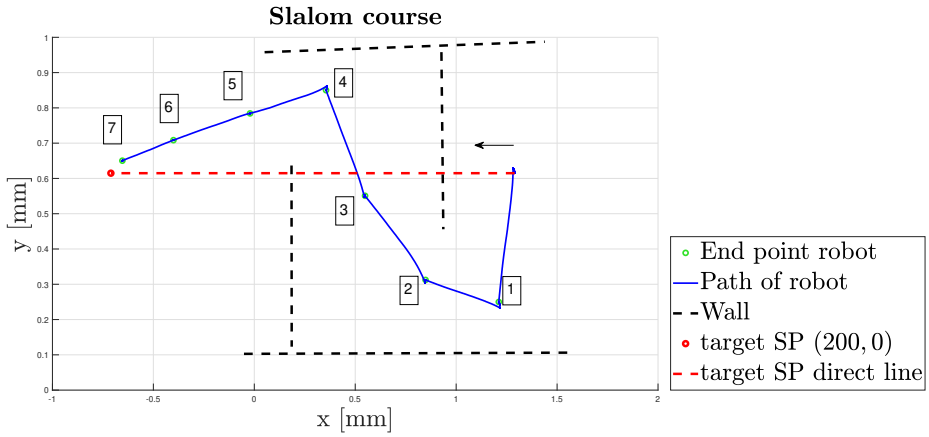


Figure 7.12: Running the path planner procedure for finding the path around the Slalom course

.

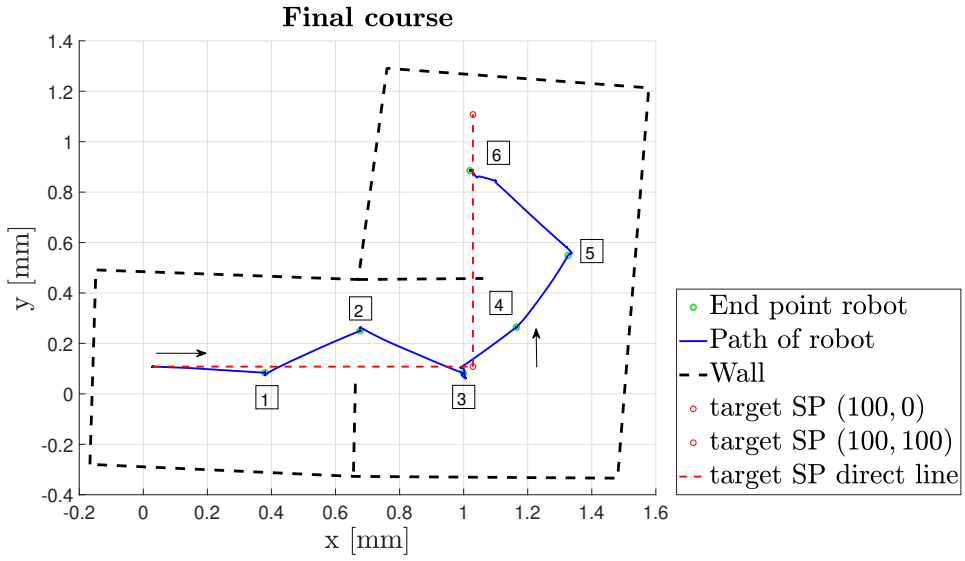


Figure 7.13: Running the path planner procedure for the Final course

.

fig. 7.11 shows the result of the path planner procedure for finding a safe path to the target coordinate in the Box course. The red dotted line visualize the direct path the robot would have taken. The robot would obvious fail to reach the target coordinate and collide with the wall represented by the black dotted line. The blue sold line shows the traversed path using the path planner procedure. As shown in fig. 7.11, the traversal of the Box course required seven augmented target set-point coordinates represented by the green dots. The Box course can be considered the easiest of the three courses, as the box is surrounded by open space. However, all of the feature in described in section 7.2 plays a part for computing a "intelligent" target set-point coordinates. As seen from the augmented target coordinate *4* and beyond, the robot is guided into the box, showing how the angle weight feature Θ is favored.

fig. 7.12 shows the result of the more advanced Slalom course. As with the box course in 7.11, the black dotted lines represent the walls, the red dotted line represent the direct path, the blue solid line represent the traversed path of the robot and the green dots represent the intermediate set-point coordinates produced by the path planner procedure. Again, the robot required seven augmented target set-point coordinates for safely traversing the course. In this course, the robot has to go into a restricted area, as well as, avoiding two walls before it can reach the

target coordinate. Another aspect is that the issued target set-point coordinate of $(2000, 0)$ is far beyond the maximum range of the IR sensor. As consequence, the proximity feature $\mathbf{P}$ is zero for the augmented target coordinates from *1* to *4*. Also, the robot cannot favor the angle to target feature $\mathbf{\Theta}$ in sector **2** shown in fig. 7.10(b). However, the combination of the different feature with the weights given in table 7.2 in combination with the amount of repulsion in $\theta_{aug,min}$ makes the robot successful in traversing the entire course.

fig. 7.13 shows the result of the final course, where the robot required three augmented target set-point coordinate for traversing both of the paths. The Final course can be considered the most advanced of the three. In similar fashion to the Slalom course, the robot has to navigate highly restricted areas. Despite, the designed algorithm in algorithm 9, with the given parameters in table 7.2, does produce augmented target set-point coordinates that result in the safe path for traversal of the environment. The parameters in table 7.2 plays a mayor part in the successful traversal of all the courses. The weights used for the the feature for obtaining $\mathbf{z}$ in eq. (7.7) have been found empirically based on observed behavior of the robot when it is navigating an environment. Both $K_{\theta,rep}$ and $K_{dist,rep}$ plays a part in the repulsiveness of $\theta_{aug,min}$ and $d_{aug,min}$. These have also been tuned empirically for obtaining the result. As seen from the result in the Slalom course in fig. 7.12, in sector **4**, at target coordinate *3*, the given $K_{\theta,rep}$ forces the robot to go to a coordinate close to the upper wall. The importance of the gain $K_{\theta,rep}$ is also shown in the Final course at the transition between sector **2** and **3**. At target coordinate *4*, the particular gain $K_{\theta,rep}$ result in an angle $\theta_{aug,min}$ that forces the robot to the leftmost wall.

The resolution of the scan field affects the performance of the procedure. A high enough resolution is needed such that the robot is able to resolve the shape of an object. The resolution is especially important when the robot is located near the edge of a wall with free space in front. This will provide the enough information to find a good angle scan step θ_{ssa} and the slope for providing adequate pushing of the augmented angle $\theta_{aug,min}$. However, higher resolution would mean longer scan times and more data to process. A resolution of 40 was found to give enough sampling points for the procedure to resolve small edges.

An observed issue with the procedure is in the computation of the augmented angle $\theta_{aug,min}$, where the angle scan step θ_{ssa} and the angle weight direction Δ_{awd} can cancel each other out. This means that the robot wont make the appropriate adjustment in the heading and that the path planner procedure cannot guarantee that the robot avoid objects in all cases. The suspected cause for the cancellation is in

the constant index used for calculating Δ_{awd}, as the issue has been observed when there are several dips and bumps in the scan field, in addition to be depended on the position and heading of the robot relative to several objects, with its own relative position and heading. Although this may be true, issuing sensible target set-point coordinate based on the relative position of the robot in the mapped environment, made for safe traversal.

However, the result in fig. 7.11, 7.12 and 7.13 shows that the robot is successful in navigating increasingly more difficult courses given sensible target set-point coordinates. In all of the three cases, the path planner procedure is able to find a safe path to the target coordinate using the extracted data from the IR sensors and calculating augmented set-point target coordinates.

Mapping

Two courses are used for testing the mapping capabilities of the robot, the Final course shown in fig. 7.10(c) and an easier course, called the Circle circle as shown in fig. 7.14.

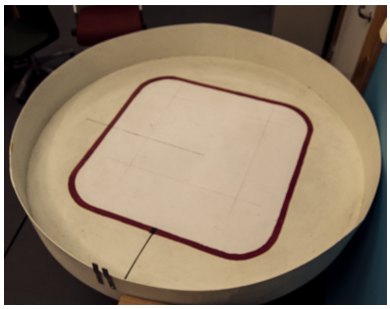
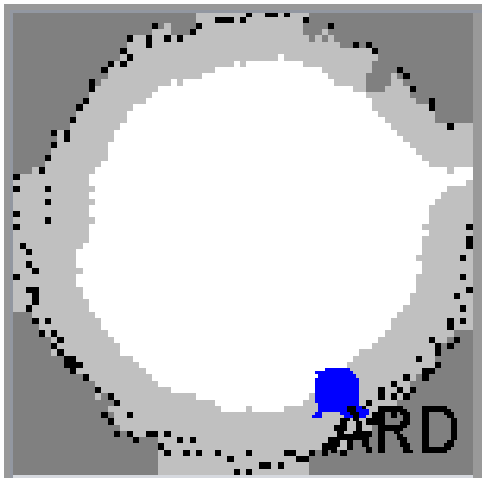


Figure 7.14: Circle course

Two mapping cases are used: First, the Path planner procedure along side the navigation procedure running on the server. And second, exclusively using the Path planner procedure for navigating and manual imputing a target set-point coordinate that is out of bound for the given course. For the manual test, the feature Θ is disabled in the Path planner procedure, meaning that the robot will follow the path with most open space.

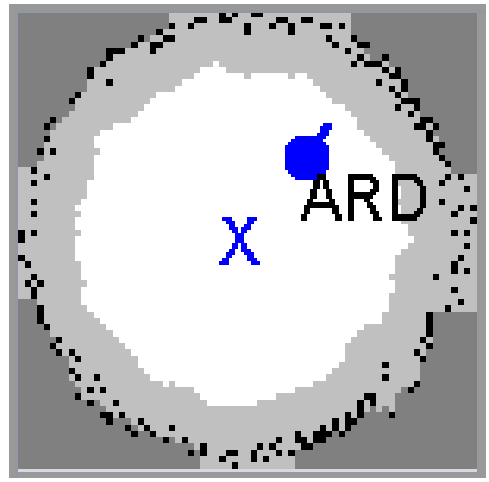
Result

fig. 7.15(a) and fig. 7.15(b) shows the result of the mapped Circle track when running the Path planner procedure with and without the server. fig. 7.16(a) and fig. 7.16(b) shows the result of the mapped Final course when running the Path planner procedure with and without the server. The robot did not fail, in the form of colliding into any object, for both courses. The result in fig. 7.15 and fig. 7.16 shows that, solely using the path planner procedure produces a similar mapping result for when it is used along side the server.



(a) Mapping of the circle course using the server in combination with the path planner procedure running on the robot

.



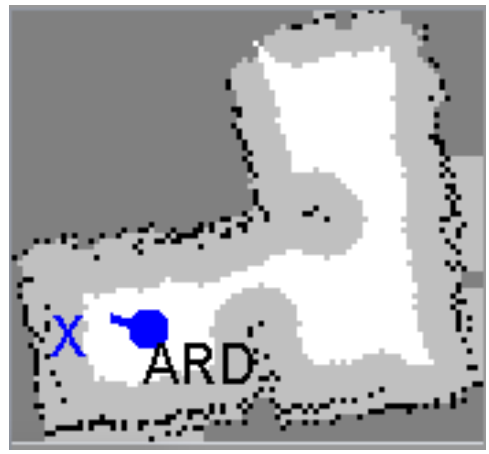
(b) Mapping of the circle track using exclusively the path planner procedure running on the robot

.

Figure 7.15: Mapping of the Circle course

(a) Mapping of the final course using the server in combination with the path planner procedure running on the robot

.



(b) Mapping of the final course using exclusively the path planner procedure running on the robot

.

Figure 7.16: Mapping of the Final course

The Path planner procedure does not produce any augmented target set-point co-

ordinates that result in any rotation over ± 90 degrees. This means that the objects in front of the robot is always considered. For the manual case, the feature Θ was disabled. Without Θ, sample points closer to the target coordinate is not favored, any path into highly restricted areas are avoided. This result in the tendency of seeking out natural opening in the environment and avoiding getting close to the walls. Another positive aspect, is that the path planner procedure is based on the relative current position and heading of the robot and produces a correspondingly relative augmented set point coordinate. This have the effect that the robot does not collide with the walls in simpler environments such as the Circle course. When testing the path planner procedure with the server in manual mode, the robot successfully mapped the Circle course for over 20 minutes.

It was observed that the mapping using the server along side the path planner procedure was slower compared to using only the path planner. A reason for this was that the collision handler running on the server seemed to conflict with the path planner procedure. An example was in the opening in sector **2** in the Final course. As described in section 7.3, for a issued target set-point coordinate, the procedure will continue to calculate augmented target set-point coordinates until the target has been reached. When the robot was in the operation of seeking out a path, the collision handler would issue a target set-point coordinate causing the robot move back to the starting point and back to the original target. Disabling the collision handler on the server, made for more efficient mapping when using the path planner procedure along side the server. This had the effect, that the server was only responsible for finding sensible target set-point coordinates based on the current mapping of the environment and the path planner procedure on the robot was responsible for finding a safe path to the target. Therefore, this mode made for the most successful and safest navigation and mapping of the environment.

Chapter 8

Discussion

During development, the Arduino IR robot have seen mayor changes and improvements. The accumulation of all the improvements described in the report is what makes the robot perform as it does. A focus have been to make the implementation memory efficient and providing good response time. The implemented code in Node 1 required 30 Kb program memory and 5 Kb data memory. Node 2 required 28 Kb program memory and 5.6 Kb data memory. table 8.1 shows the worst case running time for each task. This was found by setting a GPIO pin high at the start of the task and pulling it low at the end. The resulting pulse was measured using an oscilloscope. As seen from table 8.1 all of the task meets the deadline defined by the period in table 4.2 and table 4.3. The result also shows that the given implementation for each task gives a good system response and low interference between tasks considering the clock speed of the MCU is 16 MHz.

Task	Worst case running time
Node communication task	2.0 ms
Position estimation task	3.5 ms
Position control task	1.0 ms
Wheel speed control task	1.0 ms
Server communication task	30.0 ms
Coordinator task	150.0 μs
Path planner task	20 ms

Table 8.1: Worst case running time for each task

The Server communication task have the highest worst case running time and happens when the task transmit the handshake message. System response was favored

in the implemented if the position estimation, as seen from table 8.1. Likewise, the result in chapter 5 shows that using Kalman filtering on the gyroscope resulted in a heading error of ± 1 degree. Using the accelerometer in the position estimation resulted in an error such that the estimated position would tend to stay within a region of the real position, in comparison to the constant drifting when only using the encoders. The result from chapter 5 also indicated that the average error is in the range of 15 millimeters.

The implemented reference feed forward with state feedback LQR controller also gave a solid improvements comparing to the control system found in Arduino IR V1. However, the controller have a steady state error in the control of the position of the robot. Regardless, the deviation of 20 millimeters is almost trivial comparing to the deviation found on Arduino IR V1. The explicit feedback law in eq. (6.25) and eq. (6.26) that runs on the Atmega2560 controller is also less computationally expensive comparing to the discrete PID implementation in eq. (2.3). As the AVR instruction set on the Atmega2560 controller is based on the 8-bit RISC architecture [1], a considerable amount of instructions is needed to do computation with floating point numbers. The de-compilation of the discrete PID implementation show that it require up to 500 instructions, whereas eq. (6.25) only requires 60 instructions.

For the Path planner task, an ordinary scan take 5.0 ms, augmented scan 2.5 ms and the worst case running time is when it calculates an augmented target set-point coordinate. The worst case running time of 20 ms shows promising real-time capabilities of the procedure. The mayor limitation of the Path planner procedure is the calculation of the $\theta_{auf,min}$ using the angle weight direction Δ_{awd}. For highly dynamic scan fields where there are several peaks and valleys in the scan field, the calculated direction of $\theta_{auf,min}$ may cancel out θ_{ssa}. However, using the constant R_{dsi} for the sector size used for computing Δ_{awd} and the constant $K_{\theta,rep}$ for the angle repulsion works in most of the cases but cannot account for all. section 7.4 showed that the the path planner procedure is successfully in navigating the environment when using sensible target set-point coordinates. By disabling the collision handler on the server, combining the path planner with the server showed the best result.

Chapter 9

Further work

9.1 Further improve the position estimate

The new IMU have also a magnetometer build into it [7]. This can be used to extended the available sensors for estimating the position. Implementing multidimensional Kalman filtering for sensor fusion can be beneficial. On the contrary, the implementation need to give a significant improvements in the position estimate. In this implementation, Kalman filtering was only applied to the gyroscope as it had the most impact on the reduction of the error in the position estimate and system response was favored.

9.2 Improving the Path planner procedure

The mayor limitation with the path planner procedure is the us the constant field sector when computing Δ_{awd}. The Path planner procedure can be improved by find a way to dynamically adjust the angle weight direction Δ_{awd} or other ways to avoid canceling in $\theta_{aug,min}$. It may also be beneficial to investigate other features that is extracted from the information of the scan field or other measurements. Another improvements is to find ways to automate the tuning of the weights and investigate if the sample point and corresponding features can have separate weights.

9.3 Extended the capabilities of the Path planner procedure

Finding way for the path planner procedure to handle moving objects. As described in chapter 7, each sampled point correspond to a coordinate in the environment. It currently takes 1.2 seconds to complete a scan. This is way to slow for using the current method directly for handling moving objects. This means that scanning the environment while the robot is moving, would require saving the corresponding coordinate for each sampling while it is sampled. Another solution can be to scan faster. However, this may lead to electrical issues and the period of the task while scanning cannot be less than the time it takes to complete each

sampling. This means that faster scanning would require both hardware and software implementations.

Without considering different edge cases where, the path planner finds a coordinate that is safe to traverse. This means that each of the coordinates can be saved to form a road map from the starting point. The coordinates can be saved in the form of a stack allowing some form of depth-firs search or as a graph allowing connecting coordinates together.

9.4 Making the robot compatible with the Thread server

The Arduino IR robot need to be made compatible with the Thread server for maintaining operational status, as the Java server is in the process of being phased out in favor of the Thread server [18]. As described in section 2.1 the Thread server uses the nRF52840 dongles as a gateway for connecting to the robot. Pin connector for communicating over I2C have been added to the extension card on Node 2. The nRF52840 can be added to the robot and interfaced via I2C, as there exists available code for the gateway application [3]. The latest code developed for the nRF robot by Stenset (Master 2020) [27] can be used as a starting point.

Bibliography

[1] *Atmega2560 datasheet*. https://www.freertos.org/wp-content/uploads/2018/07/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf. Accessed: 2020-05-08.

[2] Tor Aksel N. Heirung Bjarne Foss. *Merging Optimization and Control*. Norwegian University of Science and Technology, 2016.

[3] Marius Blom. *nRF52 with OpenThread*. Accessed: 2020-08-31. 2020.

[4] John Catsoulis. *Designing Embedded Hardware*. O'Reilly, 2005.

[5] Chi-Tsong Chen. *Linear System Theory and Design*. Oxford University Press, Incorporated, 2014.

[6] F. Blanes D. Navarro G. Benet Gilabert. *Line-Based Incremental Map Building Using Infrared Sensor Ring*. Accessed: 2020-08-01. IEEE Xplore, 2008.

[7] *Datasheet ICM20948*. https://invensense.tdk.com/wp-content/uploads/2016/06/DS-000189-ICM-20948-v1.3.pdf. Accessed: 09.12.2020.

[8] Erlend Ese. *Master thesis: "Sanntidsprogrammering på samarbeidande mobilrobotar"*. Accessed: 2020-05-08. 2016.

[9] Mark W. Spong Seth Hutchinson and M. Vidyasagar. *Robot Modeling and Control*. John Wiley & Sons Inc, 2005.

[10] M. Hernández-Pajare J. Sanz Subirana J.M. Juan Zornoza. *GNSS Data Processing, Volume 1: Fundamentals and Algorithms*. Accessed: 2020-12-18. ESA Communications, 2013.

[11] Dimitris K. Manolakis John G. Proalis. *Digital Signal Processing*. Pearson, 2014.

[12] *Kalman Filter Tutorial*. https://www.kalmanfilter.net/default.aspx. Accessed: 2020-12-18.

[13] Kristian Lien. *Master thesis: "Embedded utvikling på en fjernstyrt kartleggingsrobot"*. Accessed: 2020-05-08. 2017.
[14] *lqr*. https://se.mathworks.com/help/control/ref/lqr.html. Accessed: 2020-05-08.
[15] *lqrd*. https://se.mathworks.com/help/control/ref/lqrd.html. Accessed: 2020-05-08.
[16] *LSM6DS3 datasheet*. https://www.st.com/resource/en/datasheet/lsm6ds3.pdf.
[17] *Mastering the FreeRTOS Real Time Kernel*. https://www.freertos.org/fr-content-src/uploads/2018/07/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf. Accessed: 2020-05-08.
[18] Michael Skibeli Mullins. *Implementation of SimultaneousLocalisation and Mapping in RoboticSystem using the improved RaoBlackwellized Particle Filter*. Accessed: 2020-08-31. 2020.
[19] Torgeir Myrvang. *Project: Position control of robot used for localization and mapping*. Accessed: 2020-08-31. 2020.
[20] Simen Robstad Nilssen. *Master thesis*. Accessed: 2020-05-08. 2018.
[21] *nrf51 datasheet*. https://infocenter.nordicsemi.com/pdf/nRF51822_PS_v3.1.pdf. Accessed: 2020-05-08.
[22] J.T.Gravdahl O.Egeland. *Modeling and Simulation for Automatic Control*. Marine Cybernetics, 2005.
[23] M. Pedersen. *TTK4115 Lecture State feedback*. Accessed: 2020-05-08.
[24] Andersen & Rødseth. *Master thesis*. Accessed: 2020-05-08. 2016.
[25] *SharpG2D12 datasheet*. https://global.sharp/products/device/lineup/data/pdf/datasheet/gp2y0a21yk_e.pdf. Accessed: 2020-05-08.
[26] *Source code Arduino IR V1*. Accessed: 01.01.2020.
[27] Arild Stenset. *Master: nRF52 robot with OpenThread*. Accessed: 2020-08-31. 2020.
[28] Eirik Thon. *Master thesis*. Accessed: 2020-11-08. 2016.
[29] Magne H. Johansen Tor A. Myrvoll Stefan Werner. *Estimation, Detection and classification theory*. https://learn-eu-central-1-prod-fleet01-xythos.s3-eu-central-1.amazonaws.com/5def77a38a2f7/2049630?response-content-disposition=inline%3B%20filename%2A%3DUTF-8%27%27EDC%2520-%2520Compendium.pdf&response-content-type=application%2Fpdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20200327T084821Z&X-Amz-SignedHeaders=host&X-Amz-Expires=21600&X-Amz-Credential=AKIAZH6WM4PLYI3L4QWN%2F20200327%2Feu-central-1%2Fs3%2Faws4_request&X-Amz-

Signature=a35af661fe4b82485f41c5122e1723c02948e35ea8fd92fdfce7d22
Accessed: 2020-03-26.

.1 Parameters

Various parameter used in this report and the Arduino IR V2 software.

Motor	Gear ratio	Torque
Old	120:1	$0.08\ Nm$
New	34:1	$0.118\ Nm$
Wheel	Width	Wheel radius
Old	14 mm	32.5 mm
New	26 mm	38 mm

Table 1: Specification comparison between the old and new motors

Δt	49 ms
r_w	32.5 mm
Ticks/Wheel_rev	285
K_{tw}	45
K_{vw}	0.707

Table 2: Parameters for the encoders

Gyroscope	
Low-pass filtering	119 Hz 3DbBW
Averaging filtering	$\times 8$
Output scale	$\pm 2000\ [dps]$
Sensitivity Scale Factor $F_{SF,g}$	16.4
Accelerometer	
Low-pass filtering	50 Hz 3DbBW
Averaging filtering	$\times 16$
Output scale	$\pm 2\ [g]$
Sensitivity Scale Factor $F_{SF,a}$	16384

Table 3: Configuration parameters used for the ICM 20948 IMU [7]

Δt	0.05
K_{vw}	0.707
accel scale	16384
gyro scale	16.4
gyro offset	0.635
accel mean	40
accel std	400
gyro mean	4
gyro std	4
Q	4
R	4

Table 4: Parameters used in the position estimate

m_w	0.1	$[kg]$
m_r	2.0	$[kg]$
r_w	33.5	$[mm]$
wb	198	$[mm]$
r_{wb}	99	$[mm]$
J_m	56.11	$[kg \cdot mm^2]$
J_r	6534	$[kg \cdot mm^2]$

Table 5: Physical parameters for the robot

.2 Angle mapping

To calculated the difference $\Delta\theta$ between two angles θ_1 and θ_2 in the range $[-2\pi, 2\pi]$ eq. (1) [26].

$$\Delta\theta = \begin{cases} (\theta_2 - \theta_1) - 2\pi & \theta_2 > \frac{\pi}{2} \quad \theta_1 < -\frac{\pi}{2} \\ 2\pi - (\theta_2 - \theta_1) & \theta_2 < -\frac{\pi}{2} \quad \theta_1 > \frac{\pi}{2} \\ (\theta_2 - \theta_1) & -\frac{\pi}{2} \leq \theta_2 \leq \frac{\pi}{2} \quad -\frac{\pi}{2} \leq \theta_1 \leq \frac{\pi}{2} \end{cases} \tag{1}$$

eq. (2) can be used to limit an angle in the range $[-\pi, \pi]$ [26].

$$\theta = \begin{cases} -(2\pi - \theta) & \theta > \pi \\ \theta + 2\pi & \theta < -\pi \\ \theta & -\pi \leq \theta \leq \pi \end{cases} \tag{2}$$

.3 Kalman Filtering

The Kalman filter predict the state vector $\mathbf{x}$ in four steps defined in section .3, section .3, section .3, section .3 [12] [10].

$$
\begin{aligned}
1: \quad \mathbf{K}_k &= \mathbf{P}_{k-1}\mathbf{H}^T(\mathbf{H}\mathbf{P}_{k-1}\mathbf{H}^T + \mathbf{R})^(-1) && (3)\\
2: \quad \hat{\mathbf{x}}_k &= \hat{\mathbf{x}}_{k-1} + \mathbf{K}_k(\mathbf{y}_k - \mathbf{H}\hat{\mathbf{x}}_{k-1}) && (4)\\
3: \quad \mathbf{P}_k &= (I - \hat{\mathbf{K}}_k\hat{\mathbf{H}}_k)\hat{\mathbf{P}}_k(I - \hat{\mathbf{K}}_k\hat{\mathbf{H}}_k)^T + \hat{\mathbf{K}}_k\hat{\mathbf{R}_{\mathbf{v}}}_k\hat{\mathbf{K}}_k^T && (5)\\
4: \quad \hat{\mathbf{x}}_{k+1} &= \hat{\mathbf{F}}_k\hat{\mathbf{x}}_k + \hat{\mathbf{G}}_k\hat{\mathbf{u}}_k && (6)\\
\hat{\mathbf{P}}_{k+1} &= \hat{\mathbf{F}}_k\hat{\mathbf{P}}_k\hat{\mathbf{F}}_k^T + \hat{\mathbf{Q}_{\mathbf{w}}} && (7)
\end{aligned}
$$

$\hat{\mathbf{x}}_k, \hat{\mathbf{x}}_{k-1}$ is the previous and current estimate of the state. $\hat{\mathbf{x}}_{k+1}$ is the predicted estimate. $\mathbf{y}_k$ is the measurement, $\mathbf{K}_k$ is the Kalman Gain, $\hat{\mathbf{F}}$ is the state transition matrix and $\mathbf{H}$ is observation matrix. $\hat{\mathbf{P}}, \mathbf{k}$ is used as a priory estimate uncertainty and is updated for each iteration after a prediction is made. $\mathbf{R}$ is the measurement uncertainty and $\hat{\mathbf{Q}_{\mathbf{w}}}$ is the co-variance matrix [12].

.4 Euler-Lagrange equation

The Euler-Lagrange equation is a partial differential equations for describing the dynamics of a mechanical system subject to holonomic constraints [9]. The equations makes use of generalized coordinates q and the the Lagrangian $\mathcal{L}$ defined as

$$\mathcal{L} = \mathcal{K} - \mathcal{P} \tag{8}$$

which is the difference between the kinetic and potential of a mechanical system. The Euler-Lagrange equation is defined by eq. (9) [22] [9].

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right) - \frac{\partial \mathcal{L}}{\partial q_i} = \tau_i, \quad i = 1, \ldots, n \tag{9}$$

.5 Controllability matrix

A linear time invariant system is said to be controllable if the controllability matrix $\mathcal{C}$ have full rank [5]. The matrix $\mathcal{C}$ is defined as

$$\mathcal{C} = [\mathbf{B}\ \mathbf{AB}\ \mathbf{A}^2\mathbf{B}\ \ldots\ \mathbf{A}^{n-1}\mathbf{B}] \tag{10}$$

.6 Atan2

The atan2 function is used to find the correct angle in the range $[-\pi\pi]$ between a point and a fixed coordinate systems x-axis defined by eq. (11) [9] [19] for

$\forall(x, y) \neq 0$.

$$atan2(x, y) = \begin{cases} \arctan(\frac{y}{x}) & \text{if} x > 0 \\ \arctan(\frac{y}{x}) + \pi & \text{if} x < 0 \ \text{and} \ y \geq 0 \\ \arctan(\frac{y}{x}) - \pi & \text{if} x < 0 \ \text{and} \ y < 0 \\ \frac{\pi}{2} & \text{if} x = 0 \ \text{and} \ y > 0 \\ -\frac{\pi}{2} & \text{if} x = 0 \ \text{and} \ y < 0 \end{cases} \tag{11}$$

.7 Kinematic equation

The homogeneous transformation matrix defined by eq. (12) can be used for representing position and orientation between two or more rigid bodies [9].

$$H_n^0 = \begin{bmatrix} R_n^0 & o_n^0 \\ 0 & 1 \end{bmatrix} \tag{12}$$

where R_n^0 is the rotation matrix and o_n^0 a position translation. The position a point in the n frame can be represented by eq. (13).

$$p_n^0 = R_n^0 p^n + o_n^0 \tag{13}$$

For rotation around a fixed z axis, the rotation matrix $R_{z,\theta}$ is defined by section .7.

$$R_{z,\theta} = \begin{bmatrix} c_\theta & -s_\theta & 0 \\ s_\theta & c_\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$